Proof of serializability for semistructured databases

Technical Report UA 04-01

Jan Hidders
Jan Paredaens
Roel Vercammen

University of Antwerp
Dept. Math and Computer Science
Middelheimlaan 1, BE-2020 Antwerp, Belgium
Tel. +32-3-2653-873, Fax +32-3-2653-777
{jan.hidders, jan.paredaens, roel.vercammen}@ua.ac.be

Abstract

Semistructured databases require tailor-made concurrency control mechanisms since traditional solutions for the relational model have been shown to be inadequate. Such mechanisms need to take full advantage of the hierarchical structure of semistructured data, for instance allowing concurrent updates of subtrees of, or even individual elements in, XML documents. We present a general framework to study concurrency control. This framework is document-independent in the sense that two schedules of semistructured transactions are equivalent if they are equivalent on all possible documents. We prove that it is decidable in polynomial time and space whether two given schedules in this framework are equivalent. This also solves the serializability for semistructured schedules polynomially in the number of actions and exponentially in the number of transactions.

Contents

1 Introduction **2**

2 Data Model and Operations **3**

3 Correctness of Queryless schedules **7**
3.1 Addition and Deletion Sets 8
3.2 Basic-Input-Trees and Basic-Output-Trees 9
3.3 C-Condition 17

4 Equivalence and Serializability of QL Schedules **24**
4.1 Deciding Equivalence 24
4.2 Composing Correct Schedules 26
4.3 Deciding Serializability 30

5 Equivalence and Serializability of Schedules **32**
5.1 SOP - Set Of Prefixes 33
5.2 PQRN - Potential Query Result Nodes 37
5.3 Deciding Serializability 40

References **45**

Chapter 1

Introduction

In general two actions, on two different nodes of a document tree, that are completely 'independent' from each other, cannot cause a conflict, even if they are updates. Changing the spelling of the name of one of the authors of a book and adding a chapter to the book cannot cause a conflict for instance. This consideration is the main reason why the relational approach seems to be inadequate as a concurrency control mechanism for semistructured data.

The total behavior of the processes that we consider in this paper is straightforward: each cooperating process produces a transaction of atomic actions that are queries or updates on the actual document. The transactions are interleaved by the scheduler and the resulting schedule has to be equivalent with a serial schedule. Two schedules on the same set of transactions are called equivalent if **for each possible input document** they represent the same transformation and each query gives the same result in both schedules.
The updates that we consider are very primitive: the addition of an edge of the document tree and the deletion of an edge. Semantically the addition is only defined if the added edge does not already exist in the document tree. Analogously the deletion is only defined if the deleted edge exists. A more general semantics, that does not include this constraint, can be easily simulated by adding first some queries.
There are some schedules that are not defined for any document tree. These schedules are meaningless and are called incorrect. We prove that the correctness of schedules is polynomially decidable.
In order to tackle the equivalence of schedules and transactions we first consider schedules without queries, and as such we have only to focus on the transformational behavior of the schedules. We will see that, contrary to the relational model, the swapping of the actions cannot help us in detecting the equivalence of two schedules. We prove that the equivalence of queryless schedules is also polynomially decidable and that the serializability can be decided polynomially in the size of the schedule and exponentially in the number of transactions.
Finally we generalize the results above for general schedules over the same set of transactions.
The paper is structured as follows: Section 2 defines the data model, the operations and the semistructured schedules. Section 3 studies the correctness of schedules without queries. In Section 4 we study the equivalence and the serializability for these queryless schedules. In Section 5 we generalize these results for correct schedules.

Chapter 2

Data Model and Operations

The data model we use is derived from the classical data model for semistructured data [1]. We consider directed, unordered trees in which the edges are labelled.

Consider a fixed universal set of nodes $\mathcal{N}$ and a fixed universal set of edge labels $\mathcal{L}$ not containing the symbol /.

Definition 1 *A* graph *is a tuple* (N, E) *with* $N \subseteq \mathcal{N}$ *and* $E \subseteq N \times \mathcal{L} \times N$*. A* document tree *(*DT*)* T *is a tuple* (N, E, r) *such that* (N, E) *is a graph that represents a tree with root* r*. The edges are directed from the parent to the child.*

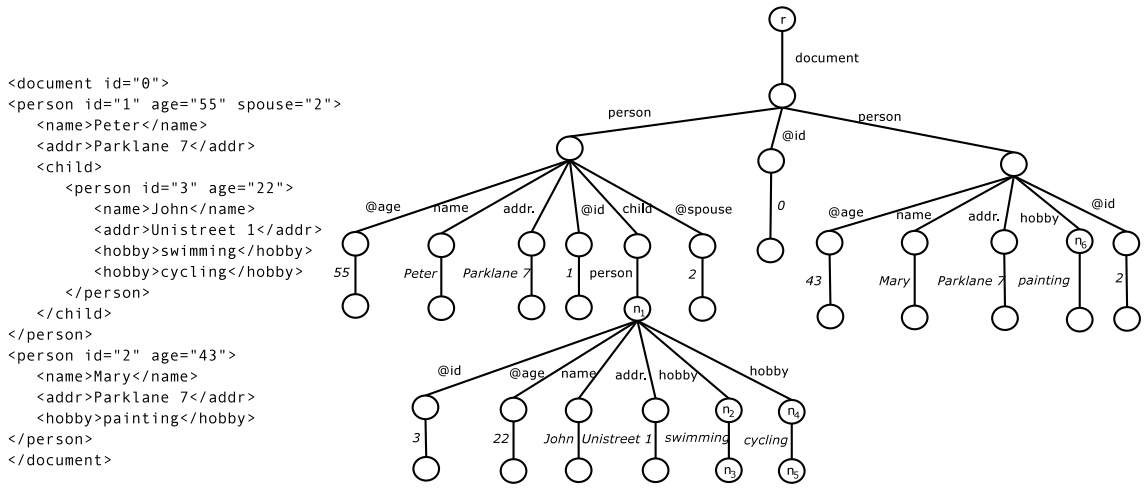

Figure 2.1: A fragment of an XML document and its DT representation.

Example 1 *Figure 2.1 shows a fragment of an XML document and its* DT *representation.*

This data model closely mimics the XML data model as illustrated in Example 1. We remark however the following differences:

- **order** Siblings are not ordered. This is not crucial, as an ordering can be simulated by using a skewed binary DT.

- **attributes** Attributes, like elements, are represented by edges labeled by the name of the attributes (started with a @) or elements. The difference is that in this data model an element may contain several attributes of the same name.
- **labels** Labels represent tag names, attribute names, values and text. References are treated as in XML: the attributes in which they appear are modeled as ordinary edges in the tree T.
- **text** Unlike in XML, it is possible for several text edges to be adjacent to each other.

A *label path* is a string of the form $l_1/\ldots/l_m$ with $m \geq 0$ and every l_i an edge label in $\mathcal{L}$. Given a path $p = ((n_1, l_1, n_2), \ldots, (n_m, l_m, n_{m+1}))$ in a graph G, the *label path of* p, denoted $\bar{\lambda}_T(p)$ (or $\bar{\lambda}(p)$ when T is subsumed) is the string $l_1/\ldots/l_m$.
Processes working on document trees do so in the context of a general programming language that includes an interface to a document server which manages transactions on documents. The process generates a list of operations that will access the document. In general there are three types of operations: the query, the addition and the deletion. The input to a query-operation will be a node and a path expression, while the result of the invocation of a query-operation will be a set of nodes. The programming language includes the concepts of sets, and has constructs to iterate over their entire contents. The input to an addition or a deletion will be an edge. The result of an addition or a deletion will be a simple transformation of the original tree into a new tree. If the result would not be a tree anymore it is not defined.
We now define the path expressions and the query-operations, subsuming a given DT T.
The syntax of path expressions[1] is given by $\mathcal{P}$:

$$\begin{aligned} \mathcal{P} &::= pe_\epsilon \mid \mathcal{P}^+ \\ \mathcal{P}^+ &::= \mathcal{F} \mid \mathcal{P}^+/\mathcal{F} \mid \mathcal{P}^+//\mathcal{F} \\ \mathcal{F} &::= * \mid \mathcal{L} \end{aligned}$$

The set $\mathbf{L}(pe)$ of label paths represented by a path expression pe is defined as follows:

$$\begin{aligned} \mathbf{L}(pe_\epsilon) &= \{\epsilon\} \\ \mathbf{L}(*) &= \mathcal{L} \\ \mathbf{L}(l) &= \{l\} \\ \mathbf{L}(pe/f) &= \mathbf{L}(pe) \cdot \{/\} \cdot \mathbf{L}(f) \\ \mathbf{L}(pe//f) &= \mathbf{L}(pe) \cdot \{/\} \cdot (\mathcal{L} \cdot \{/\})^* \cdot \mathbf{L}(f) \end{aligned}$$

We will first give some examples of path expressions and their languages.

Example 2 *Some legal path expressions are a/b, $a//*$ and $*//*$. Examples of illegal path expressions are $a/$, $a//$, $/a$ and a/pe_ϵ.*
Suppose $\mathcal{L} = \{a, b, c\}$. Then for example $\mathbf{L}(a//*)$= $\{a/a$, a/b, a/c, $a/a/a$, $a/a/b$, $a/a/c$, $a/b/a$, $a/b/b$, $a/b/c$,...$\}$. Note also that different path expressions may have the same language. For example $\mathbf{L}(*//*/*) = \mathbf{L}(*/*//*) = \mathbf{L}(*/*/*) \cup \mathbf{L}(*/*/*//*)$.

Let n be an arbitrary node of T and pe a path expression. We now define the three kinds of operations: the query, the addition and the deletion.

[1] Remark that path expressions form a subset of XPath expressions.

Definition 2 *The query-operation* $\text{query}(n, pe)$ *returns a set of nodes, and is defined as follows:*

- $\text{query}(n, pe)$ *with* $n \in \mathcal{N}$ *and* $pe \in \mathcal{P}$*. The result of a query on a* DT T *is defined as* $\text{query}(n, pe)[T] = \{n' \in N \mid \exists p$ a simple path in T from n to n' with $\bar{\lambda}(p) \in \mathbf{L}(pe)\}$.

The update operations $\text{add}(n, l, n')$ *and* $\text{del}(n, l, n')$ *return no value but transform a* DT $T = (N, E, r)$ *into a new* DT $T' = (N', E', r)$*:*

- $\text{add}(n, l, n')$ *with* $n, n' \in \mathcal{N}$ *and* $l \in \mathcal{L}$*. The resulting* $T' = \text{add}(n, l, n')[T]$ *is defined by* $E' = E \cup \{(n, l, n')\}$ *and* $N' = N \cup \{n'\}$*. If the resulting* T' *is not a document tree anymore or* (n, l, n') *was already in the document tree then the operation is undefined.*
- $\text{del}(n, l, n')$ *with* $n, n' \in \mathcal{N}$ *and* $l \in \mathcal{L}$*. The resulting* $T' = \text{del}(n, l, n')[T]$ *is defined by* $E' = E - \{(n, l, n')\}$ *and* $N' = N - \{n'\}$*. If the resulting* T' *is not a document tree anymore or* (n, l, n') *was not in the document tree then the operation is undefined.*

Example 3 *Suppose that the following sequence of update operations is applied on the document tree of Figure 2.1.*

$\text{del}(n_2,$ "`swimming`"$, n_3)$
$\text{del}(n_1,$ "`hobby`"$, n_2)$
$\text{del}(n_4,$ "`cycling`"$, n_5)$
$\text{add}(n_4,$ "`triathlon`"$, n_5)$

This results in the new document tree of Figure 2.2. The execution of the query operation $query(r, //hobby)$ *will result in the nodes in which an edge with label "hobby" ends. Hence the execution of this query on the document tree of Figure 2.2 has the result set* $\{n_4, n_6\}$*.*

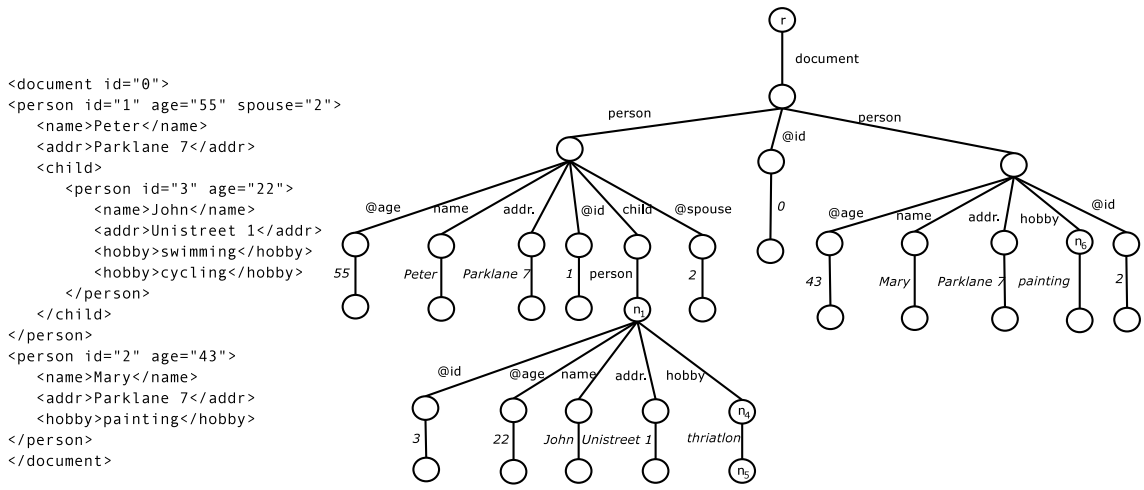

Figure 2.2: A fragment of an XML document and its DT representation.

We now give some straightforward definitions of schedules and their semantics.

Definition 3 *An* action *is a pair* (o, t)*, where* o *is one of the three operations* query(n, pe), add(n, l, n') *and* del(n, l, n') *and* t *is a transaction identifier. A* transaction *is a sequence of actions with the same transaction identifier. A* schedule *over a set of transactions is an interleaving of these transactions. The size* n_S *of a schedule* S *is the length of its straightforward encoding on a Turing tape*[2]*. The length* n_a *of a schedule is the number of actions that appear in the schedule.*
We can apply *a schedule* S *on a* DT T*. The result of such application is*

- *the* DT *that results from the sequential application of the actions of* S*; this* DT *is denoted by* $S[T]$
- *for each query in* S*, the result of this query.*

If some of these actions are undefined the application is undefined. An input document tree T *is a* DT *for which the application of* $S[T]$ *is defined. An output document tree* T *is a* DT *such that there exists a input document tree* T' *for which the application of* $S[T']$ *equals* T*.*
Two schedules are equivalent on a DT T *iff their application on* T *has the same result. Two schedules are equivalent iff they are defined on the same non-empty set of* DT*'s and they are equivalent on these* DT*'s. The definition of serial and serializable schedules is straightforward.*

Since a transaction is a special case of a schedule all the definitions on schedules also apply on transactions.

Note that the equivalence of schedules and transactions is a document-independent definition.
Let
$T_1 = (\{n_1, n_2\}, \{(n_1, b, n_2)\}, n_1)$,
$T_2 = (\{n_1, n_2\}, \{(n_1, a, n_2)\}, n_1)$ and
$T_3 = (\{n_1\}, \emptyset, n_1)$ be three DT's and let
$S_1 = (\text{add}(n_2, b, n_3), t_1), (\text{query}(n_1, a/b), t_2)$ and
$S_2 = (\text{query}(n_1, a/b), t_2), (\text{add}(n_2, b, n_3), t_1)$ be two schedules.
S_1 and S_2 are equivalent on T_1, they are not equivalent on T_2 and their application is undefined on T_3.
Let S_3 be the empty schedule and
$S_4 = (\text{add}(n_1, l_1, n_2), t_1), (\text{del}(n_1, l_1, n_2), t_2)$.
S_3 and S_4 are not equivalent although they are equivalent on many DT's. S_4 is not defined on any DT with edge (n_1, l_1, n_2), while S_3 is defined on all DT's.

[2]We assume that nodes can be encoded in $O(1)$-space

Chapter 3

Correctness of Queryless schedules

A schedule is called *queryless* (QL) iff it contains no queries. If the first occurence of the node n (the edge (m, l, n)) in a QL schedule S has the form of the operator o, then we say that $\phi_S(n, o)$ ($\phi_S((m, l, n), o)$) holds. Else $\phi_S(n, o)$ does not hold. [1] Analogously, $\lambda_S(n, o)$ ($\lambda_S((m, l, n), o)$) indicates that the last occurrence of the node n (the edge (m, l, n)) in the QL schedule S has the form of the operation o.

Because of the way that operations can fail it is possible that the application of a certain transaction is not defined for any document tree. We are not interested in such transactions. We call a transaction t *correct* iff there is a DT T with $t[T]$ defined.

Example 4 *The next transaction is correct:*
$(\text{add}(r, l_1, n_1), t_1)$, $(\text{del}(r, l_1, n_1), t_1)$,
$(\text{add}(r, l_2, n_2), t_1)$, $(\text{del}(r, l_2, n_2), t_1)$,
$(\text{add}(r, l_2, n_2), t_1)$, $(\text{del}(r, l_2, n_2), t_1)$.
The next transaction is not correct:
$(\text{add}(n_1, l_1, n), t_1)$, $(\text{add}(n_2, l_2, n), t_1)$.

We call a schedule S *correct* iff there is a DT T with $S[T]$ defined. But there are correct schedules that cannot be serializable because they contain an incorrect transaction. For instance the correct schedule
$S = (\text{add}(r, l_1, n_1), t_1), (\text{del}(r, l_1, n_1), t_2), (\text{add}(r, l_1, n_1), t_1)$
is defined on $T = (\{r\}, \emptyset, r)$ but that is not serializable because of the transaction t_1 that is not correct. Every equivalent serial QL schedule would be undefined! Transaction t_1 has the property that all QL schedules over a set of transactions that contain t_1 are non-serializable, independent of T.

Note that the definition of correct QL schedule is document-independent. It is clear that we are only interested in correct transactions and schedules. Remark also that if two QL schedules are equivalent then they are both correct, because in order for two QL schedules to be equivalent, they need to be defined on a non-empty set of DTs (definition 3). This equivalence relation is defined on the set of correct QL schedules.

[1] For example, $\phi_S(n_2, \text{add}(r, l_2, n_2))$ in the correct QL schedule in Example 4.

3.1 Addition and Deletion Sets

By ADD(S) we denote the set of edges that are added by the QL schedule S, i.e., they are added without being removed again afterwards, and by DEL(S) we denote the set of edges that are deleted by the QL schedule S, i.e., they are deleted without being added again afterwards. We now first give a formal definition of ADD and DEL and then prove that this corresponds to the informal notion previously described.

Definition 4 *Let S be a correct QL schedule. We denote*
$\mathrm{ADD}(S) = \{(m,l,n) \mid \lambda_S((m,l,n), \mathrm{add}(m,l,n))\}$
$\mathrm{DEL}(S) = \{(m,l,n) \mid \lambda_S((m,l,n), \mathrm{del}(m,l,n))\}$
We call ADD(S) *the addition set of S and* DEL(S) *its deletion set.*

Remark that two correct QL schedules with the same ADD and DEL are not necessarily equivalent. Indeed $S_1 = (\mathrm{del}(n_1,l_1,n_2),t_2)$ and $S_2 = (\mathrm{add}(n_1,l_1,n_2),t_1)(\mathrm{del}(n_1,l_1,n_2),t_2)$ are not equivalent although $\mathrm{ADD}(S_1) = \mathrm{ADD}(S_2)$ and $\mathrm{DEL}(S_1) = \mathrm{DEL}(S_2)$. Furthermore, note that $\mathrm{ADD}(S) \cap \mathrm{DEL}(S) = \emptyset$, since there is at most one last action concerning a given edge (m,l,n), hence both $\lambda_S((m,l,n), \mathrm{add}(m,l,n))$ and $\lambda_S((m,l,n), \mathrm{del}(m,l,n))$ cannot hold for the same schedule S.

Let T be a DT and E be a set of edges. We denote by $T \cup E$ the graph obtained by adding the edges of E to T and by $T - E$ the graph obtained by deleting the edges of E from T. Note that $T \cup E$ nor $T - E$ are necessarily DT's.

Lemma 1 *Let S be a correct QL schedule and T a document tree for which $S[T]$ is defined. Then $S[T] = T \cup \mathrm{ADD}(S) - \mathrm{DEL}(S)$.*

Proof We prove this lemma by induction on the number of actions n_a of the schedule S. If $n_a = 0$ then S is the empty schedule, $\mathrm{ADD}(S) = \emptyset$ and $\mathrm{DEL}(S) = \emptyset$, so $S[T] = T \cup \mathrm{ADD}(S) - \mathrm{DEL}(S) = T$. Suppose that the lemma holds for $n_a < K$.
Let S be a schedule of length $n_a = K$ and S' the same schedule without the first operation o of S. If the result of $o[T] = T'$, then we know by the induction hypothesis (i.h.) that $S'[T'] = T' \cup \mathrm{ADD}(S') - \mathrm{DEL}(S')$. Suppose o acts on the edge (m,l,n). We now have two possibilities:

- If $(m,l,n) \in \mathrm{DEL}(S')$ or $(m,l,n) \in \mathrm{ADD}(S')$, then the addition and deletion sets of both S and S' are the same, since o is not the last operation on (m,l,n).
  - If $o = \mathrm{add}((m,l,n),t_i)$ then $o[T] = T \cup \{(m,l,n)\}$.
    $S[T] = S'[o[T]]$
    $= (T \cup \{(m,l,n)\}) \cup \mathrm{ADD}(S') - \mathrm{DEL}(S')$
    $= (T \cup \{(m,l,n)\}) \cup \mathrm{ADD}(S) - \mathrm{DEL}(S)$
    $= T \cup \mathrm{ADD}(S) - \mathrm{DEL}(S)$
  - If $o = \mathrm{del}((m,l,n),t_i)$ then $o[T] = T - \{(m,l,n)\}$.
    $S[T] = S'[o[T]]$
    $= T - \{(m,l,n)\} \cup \mathrm{ADD}(S') - \mathrm{DEL}(S')$
    $= T - \{(m,l,n)\} \cup \mathrm{ADD}(S) - \mathrm{DEL}(S)$
    $= T \cup \mathrm{ADD}(S) - \mathrm{DEL}(S)$

It is important to see the last simplification is only possible since (m, l, n) is in $\text{ADD}(S)$ or in $\text{DEL}(S)$.

- If $(m, l, n) \notin \text{DEL}(S')$ and $(m, l, n) \notin \text{ADD}(S')$, then o is the last operation on (m, l, n) in the schedule S.
  - If $o = \text{add}((m, l, n), t_i)$ then $o[T] = T \cup \{(m, l, n)\}$, $\text{ADD}(S) = \text{ADD}(S') \cup (m, l, n)$ and $\text{DEL}(S) = \text{DEL}(S')$.
    $$\begin{aligned} S[T] &= S'[o[T]] \\ &= T \cup \{(m, l, n)\} \cup \text{ADD}(S') - \text{DEL}(S') \\ &= T \cup (\{(m, l, n)\} \cup \text{ADD}(S')) - \text{DEL}(S) \\ &= T \cup \text{ADD}(S) - \text{DEL}(S) \end{aligned}$$
  - If $o = \text{del}((m, l, n), t_i)$ then $o[T] = T - \{(m, l, n)\}$, $\text{ADD}(S) = \text{ADD}(S')$ and $\text{DEL}(S) = \text{DEL}(S') \cup (m, l, n)$
    $$\begin{aligned} S[T] &= S'[o[T]] \\ &= (T - \{(m, l, n)\}) \cup \text{ADD}(S') - \text{DEL}(S') \\ &= (T \cup \text{ADD}(S) - \{(m, l, n)\}) - \text{DEL}(S') \\ &= (T \cup \text{ADD}(S)) - ((\{(m, l, n)\} \cup \text{DEL}(S'))) \\ &= T \cup \text{ADD}(S) - \text{DEL}(S) \end{aligned}$$

■

3.2 Basic-Input-Trees and Basic-Output-Trees

We will characterize correct QL schedules and prove that this property is decidable. For this purpose we will first attempt to characterize for which document trees a given correct QL schedule S is defined, and what the properties are of the document trees that result from a QL schedule. We do this by defining the sets $N_I^{min}(S)$, $N_I^{max}(S)$, $E_I^{min}(S)$ and $E_I^{max}(S)$, whose informal meaning is respectively the set of nodes that are required in the input document trees on which S is defined, the set of nodes that are allowed, the set of edges that are required and the set of edges that are allowed. In the same way we define the sets $N_O^{min}(S)$, $N_O^{max}(S)$, $E_O^{min}(S)$ and $E_O^{max}(S)$, whose informal meaning is respectively the set of nodes that are required in an output document tree of S, the set of nodes that are allowed, the set of edges that are required and the set of edges that are allowed.

Definition 5 *Let S be a QL schedule. We define the sets $N_I^{min}(S)$, $N_I^{max}(S)$, $E_I^{min}(S)$ and $E_I^{max}(S)$, and the sets $N_O^{min}(S)$, $N_O^{max}(S)$, $E_O^{min}(S)$ and $E_O^{max}(S)$ as in Figure 3.1.*
A DT *T is called a basic-input-tree (basic-output-tree) of S iff it contains all the nodes of $N_I^{min}(S)$ ($N_O^{min}(S)$), only nodes of $N_I^{max}(S)$ ($N_O^{max}(S)$), all the edges of $E_I^{min}(S)$ ($E_O^{min}(S)$) and only edges of $E_I^{max}(S)$ ($E_O^{max}(S)$).*

$$
\begin{array}{l}
N_I^{min}(S) = \{m \mid \phi_S(m, \text{add}(m,l,n))\} \cup \{m \mid \phi_S(m, \text{del}(m,l,n))\} \ \cup \{n \mid \phi_S(n, \text{del}(m,l,n))\} \\
N_I^{max}(S) = \mathcal{N} - \{n \mid \phi_S(n, \text{add}(m,l,n))\} \\
E_I^{min}(S) \ = \{(m,l,n) \mid \ \phi_S((m,l,n), \text{del}(m,l,n))\} \\
E_I^{max}(S) = E_I^{min}(S) \ \cup \ \{(m,l,n) \mid \ \text{no } (m_1,l_1,m) \text{ nor } (m_1,l_1,n) \text{ occurs in } S\} \\
N_O^{min}(S) = \{m \mid \lambda_S(m, \text{del}(m,l,n))\} \cup \{m \mid \lambda_S(m, \text{add}(m,l,n))\} \cup \{n \mid \lambda_S(n, \text{add}(m,l,n))\} \\
N_O^{max}(S) = \mathcal{N} - \{n \mid \lambda_S(n, \text{del}(m,l,n))\} \\
E_O^{min}(S) \ = \{(m,l,n) \mid \ \lambda_S((m,l,n), \text{add}(m,l,n))\} \\
E_O^{max}(S) = E_O^{min}(S) \cup \ \{(m,l,n) \mid \ \text{no } (m_1,l_1,m) \text{ nor } (m_1,l_1,n) \text{ occurs in } S\}
\end{array}
$$

Figure 3.1: The Definition of the Basic Input and Output Sets.

Consider $S = (\text{add}(n_1,l_1,n_2),t_1),\ (\text{del}(n_4,l_2,n_3),t_2),\ (\text{del}(n_1,l_1,n_4),t_3)$ then

$$
\begin{array}{lcl}
N_I^{min}(S) & = & \{n_1,n_3,n_4\} \\
N_I^{max}(S) & = & \mathcal{N} - \{n_2\} \\
E_I^{min}(S) & = & \{(n_4,l_2,n_3),(n_1,l_1,n_4)\} \\
E_I^{max}(S) & = & E_I^{min}(S) \ \cup \{(m,l,n) \in \mathcal{N} \times \mathcal{L} \times \mathcal{N} \mid m,n \neq n_2,n_3,n_4\} \\
N_O^{min}(S) & = & \{n_1,n_2\} \\
N_O^{max}(S) & = & \mathcal{N} - \{n_3,n_4\} \\
E_O^{min}(S) & = & \{(n_1,l_1,n_2)\} \\
E_O^{max}(S) & = & E_O^{min}(S) \ \cup \{(m,l,n) \in \mathcal{N} \times \mathcal{L} \times \mathcal{N} \mid m,n \neq n_2,n_3,n_4\}
\end{array}
$$

We will show that the informal meaning for the E- and N-sets is correct in respect to their definition. In order to prove this, we first show that if a node does not appear in the schedule, then it cannot be required or disallowed and if an edge does not appear in the schedule S, then it cannot be required. For an edge (m,l,n), it does not suffice to say that (m,l,n) has to appear in S to be disallowed. For example the schedule with only operation $\text{add}((m',l',n),t_i)$ would disallow (m,l,n) to be in the input document tree.

Lemma 2 *Let S be a correct QL schedule. For any* DT *T, denote N_T as the set of nodes of T and E_T as the set of edges of T. Then the following holds:*

- *If $((\forall T).(S[T]$ defined $\rightarrow n \in N_T))$ then n appears in an action of S.*
- *If $((\forall T).(n \in N_T \rightarrow S[T]$ undefined$))$ then n appears in an action of S.*
- *If $((\forall T).(S[T]$ defined $\rightarrow (m,l,n) \in E_T))$ then (m,l,n) appears in an action of S.*
- *If $((\forall T).((m,l,n) \in E_T \rightarrow S[T]$ undefined$))$ then an edge of the form (m_1,l_1,m) or (m_1,l_1,n) appears in an action of S.*

Proof. We prove the four implications seperately.

- Let $(\forall T).(S[T]$ defined $\rightarrow n \in N_T)$. Suppose that n does not appear in an action of S. Since S is correct, we know $(\exists T).(S[T]$ defined$)$, for instance T_1. Clearly $n \in N_{T_1}$, since otherwise we get a contradiction. Since $\mathcal{N}$ is infinite, there exists a n' that is not in T_1 and not in an action of S. Let T_2 be the same DT as T_1, but with n substituted by n'. Since n does not appear in S, this renaming has no influence on the fact whether the

new document tree is defined or undefined. Hence $S[T_2]$ is defined and $n \notin N_{T_2}$. This contradicts our assumption, so n has to appear in an action of S.

- Let $(\forall T).(n \in N_T \rightarrow S[T]$ undefined). Suppose that n does not appear in an action of S. Since S is correct, we know $\exists T : S[T]$ defined. Let T_1 be such a T on which the application of S is defined. Clearly $n \notin N_{T_1}$, since otherwise we get a contradiction. From the definition of the actions follows that the root never changes. Let r_{T_1} be the root of T_1, then all operations in S act on the root r_{T_1} or its descendants. If we construct a new DT $T_2 = (N_{T_1} \cup n, E_{T_1} \cup (n, l, r_{T_1}), n)$, then clearly $S[T_2]$ is defined, since all actions occur below r_{T_1} and no action occurs on n or the edge from n to r_{T_1}. This contradicts the assumption '$(\forall T).(n \in N_T \rightarrow S[T]$ undefined)', hence n has to appear in an action of S.

- Let $(\forall T).(S[T]$ defined $\rightarrow (m, l, n) \in E_T)$. Suppose that (m, l, n) does not appear in an action of S. Since S is correct, we know $(\exists T).(S[T]$ defined). Let T_1 be such a T on which the application of S is defined. Clearly $(m, l, n) \in E_{T_1}$, since otherwise we get a contradiction. Since $\mathcal{L}$ is infinite, there exists a l' that is not used as label of an edge in an action of S. Let T_2 be the same DT as T_1, but with (m, l, n) substituted by (m, l', n). Since (m, l', n) does not appear in S, this renaming has no influence on the fact whether the new document tree is defined or undefined. Hence $S[T_2]$ is defined and $(m, l, n) \notin E_{T_2}$. This contradicts our assumption, so (m, l, n) has to appear in an action of S.

- Suppose that no (m_1, l_1, m) nor (m_1, l_1, n) appears in an action of S. We then have to show that there exists a document tree T such that $(m, l, n) \in E_T$ and $S[T]$ is defined. First we construct a candidate tree T', then we prove that S is defined on this tree.

  - Let T be the graph with edges $E_I^{min}(S)$ and nodes $N_I^{min}(S) \cup \{m, n\}$.
  - T is a forest, since S is correct and $E_I^{min}(S)$ is a subgraph of each document tree on which S is defined[2].
  - We know that only the edges of $E_I^{min}(S)$ are in T, i.e., only edges on which an action occurs. Since no edge ending in m or n appears in an action of S, no edge ending in m or n appears in T. Hence the nodes m and n are root of a tree in T.
  - Let r be a node that does not appear in S or T and $T' = (r \cup N_T, E_T \cup \{(r, l_1, n_1) | n_1$ is root of a tree in T and $n_1 \neq n\} \cup \{(m, l, n)\}, r)$ (with N_T the nodes from T and E_T the edges from T).
  - All trees of T (except the tree with root m) are connected to a new root r. The tree with root m is connected to n, hence T' is a tree.

  We now have to prove that $S[T']$ is defined. We will prove this by induction on the length of the schedule S. If S is the empty schedule then $S[T]$ is defined on all DTs and hence $S[T']$ is defined. Suppose that for all schedules S of length $n_a < K$ the application $S[T']$ is defined (induction hypothesis).
  Let S be a schedule of length $n_a = K$ and o the last action of S. Then $S = S'.o$, where S' is a schedule of length $K - 1$. From the induction hypothesis follows that $S'[T']$ is

[2]We use in the proof of this part of the lemma the result for the min-sets of Lemma 4. This is sound (i.e., we do not have a circular reasoning) since that part of Lemma 4 only uses the two first parts of this lemma.

defined and hence (by Lemma 1) $S'[T'] = T' \cup \text{ADD}(S') - \text{DEL}(S')$. Suppose that $S[T']$ is not defined. Then the last operation o must fail. We now have two possibilities:

- The operation o is an addition (i.e., $o = \text{add}((m_2, l_2, n_2), t_i)$). Then o can fail for two reasons: n_2 is already in $S'[T']$ or m_2 is not in $S'[T']$.
  Suppose n_2 is already in $T'' = S'[T']$. This node cannot be in $\text{ADD}(S')$, since it is independent of the document tree and hence the node would occur in every output document tree of S', so $S'.o = S$ would be incorrect. Therefore a node occuring in $T' - DEL(S')$ causes the conflict, but since $N_{T'} = N_I^{min}(S) \cup \{m, n, r\}$ and S is correct, the node causing the conflict must be m, n or r (otherwise S will fail on all document trees and hence be not correct). Since r is chosen not to occur in S and no (m_1, l_1, n) nor (m_1, l_1, m) occurs in S, we get a contradiction.
  Hence o will fail because m_2 is not in $S'[T']$. It's obvious that m_2 does not appear in $\text{ADD}(S')$, since these edges are not deleted afterwards and hence m_2 would be in $S'[T']]$. If m_2 is in $\text{DEL}(S')$ then the operation o in S will always fail and hence S is incorrect. Therefore m_2 is not in T' and does not appear as child node in any action of S'. But in this case the operation o is the first addition of m_2 and hence $m_2 \in N_I^{min}(S)$ and hence in T'.
  Hence o cannot fail.
- The operation o is a deletion (i.e., $o = \text{del}((m_2, l_2, n_2), t_i)$). Then o can fail for two reasons: (m_2, l_2, n_2) is not in $S'[T']$ or there is an edge (n_2, l_3, m_3) in $S'[T']$.
  Suppose o fails because (m_2, l_2, n_2) is not in $S'[T']$. It's obvious that (m_2, l_2, n_2) is not in $\text{ADD}(S')$ (otherwise the edge would be in $S'[T']$. Therefore either $(m_2, l_2, n_2) \notin E_{T'}$ or $(m_2, l_2, n_2) \in \text{DEL}(S')$. Since the second case is independent of the document tree and S is correct, (m_2, l_2, n_2) is not in $E_{T'}$ and this deletion is the first action on (m_2, l_2, n_2). But then we know from the definition of basic-input-trees that $(m_2, l_2, n_2) \in E_I^{min}(S)$ and hence in $E_{T'}$. This is a contradiction.
  Hence o will fail because there is an edge (n_2, l_3, n_3) in $S'[T']$. This edge cannot be in $\text{ADD}(S')$ since this set is independent of the document tree and the edge would occur in every output document tree of S', resulting in the application of S to be always undefined (and hence S would be incorrect). Furthermore $(n_2, l_3, n_3) \notin \text{DEL}(S')$ since then it is impossible for $S'[T']$ to contain the edge (n_2, l_3, m_3). Therefore (n_2, l_3, n_3) has to be in $E_{T'} = E_I^{min}(S') \cup \{(r, l_1, n_1) | n_1$ is root of a tree in T and $n_1 \neq m\} \cup \{(m, l, n)\}$. Since no action occurs on (n_2, l_3, n_3) in S' we know that $(n_2, l_3, n_3) \notin E_I^{min}(S')$. Furthermore $(n_2, l_3, n_3) \neq (m, l, n)$ since no action on an edge ending in n (ande hence (m, l, n)) occurs in S. Therefore (n_2, l_3, n_3) has to be in $\{(r, l_1, n_1) | n_1$ is root of a tree in T and $n_1 \neq m\}$, but since r does not occur in S, we get a contradiction.
  Hence o cannot fail.

This gives a contradiction with our assumption that $S[T']$ is undefined, since $o[S'[T']] = S[T']$ is defined.

■

Before we prove that the informal meaning of the basic input trees and basic output trees is correct, we first define the reverse of a QL schedule. This will help us in simplifying our proof.

Definition 6 *Let S be a QL schedule. S^σ, the reverse of S where every addition of an edge is substituted by the deletion of the edge and vice versa.*

Lemma 3 *S^σ is the 'undo' operation for the schedule S:*

- *If $S[T]$ is defined, then $S^\sigma[S[T]]$ is defined and $S^\sigma[S[T]] = T$.*
- *If $S^\sigma[T]$ is defined, then $S[S^\sigma[T]]$ is defined and $S[S^\sigma[T]] = T$.*
- $N_O^{min}(S) = N_I^{min}(S^\sigma)$
- $E_O^{min}(S) = E_I^{min}(S^\sigma)$
- $N_O^{max}(S) = N_I^{max}(S^\sigma)$
- $E_O^{max}(S) = E_I^{max}(S^\sigma)$

Proof. In this proof we will use the operator σ for the substition of a deletion by an addition and vice versa: $\sigma(\text{add}(m,l,n)) = \text{del}(m,l,n)$ and $\sigma(\text{del}(m,l,n)) = \text{add}(m,l,n)$. Some properties follow directly from definition 6:
$\lambda_S((m,l,n),o) \Leftrightarrow \phi_{S^\sigma}((m,l,n),\sigma(o))$
$\lambda_S(m,o) \Leftrightarrow \phi_{S^\sigma}(m,\sigma(o))$
Using these two properties, we can derive directly the four equations for the input and output sets by substituting the output sets with the input sets of the reverse operation. Hence we only have to prove that if $S[T]$ is defined, then $S^\sigma[S[T]]$ is defined and $S^\sigma[S[T]] = T$.

Suppose $S[T] = T'$ is defined. We will show that $S^\sigma[T'] = T$ and hence S^σ is defined. We know $S^\sigma[T'] = S^\sigma[S[T]] = (S.S^\sigma)[T]$. We will prove that this is defined by induction on the length n_a of the schedule S. If S is the empty schedule then S^σ is the empty schedule too and hence $S^\sigma[S[T]] = T$ is defined. Suppose that $(S.S^\sigma)[T]$ is defined for each schedule S of length $n_a < K$ (induction hypothesis). Then $S = S'.o$ is a schedule of length $n_a = K$ where o is the last operation of S. Hence $S^\sigma = (\sigma(o)).(S')^\sigma$. Let T be a tree for which $S[T]$ is defined. Then there is a document tree $T'' = S'[T]$. For this T'', the operation o is defined since otherwise $S[T]$ is undefined. In the schedule $(S.S^\sigma)$ the operation o is directly followed by the operation $\sigma(o)$. If o adds an edge, then $\sigma(o)$ deletes this edge and hence $\sigma(o)$ is defined on $o[T'']$. If o deletes an edge, then $\sigma(o)$ adds this edge and hence $\sigma(o)$ is defined on $o[T'']$. In both cases $(\sigma(o))[o[T'']] = T' = S'[T]$. Hence $S^\sigma[S[T]] = S'^\sigma[(\sigma(o))[o[S'[T]]] = S'^\sigma[S'[T]]$. From the induction hypothesis we know that this is defined and that it equals T. We now still have to prove the second implication of the lemma, but this follows from the first implication, since we know that $(S^\sigma)^\sigma = S$ (Definition 3). ∎

We now use the last three lemmas in order to prove that the informal meaning of the input-document-trees and output-document-trees is correct iff S is correct.

Lemma 4 *The definition of basic-input-trees and basic-output-trees agrees with its informal meaning, i.e., for a correct QL schedule S, the following properties hold:*
$N_I^{min}(S)$ is the set of nodes required in the input document tree of S.
$N_I^{max}(S)$ is the set of nodes allowed in the input document tree of S.
$E_I^{min}(S)$ is the set of edges required in the input document tree of S.
$E_I^{max}(S)$ is the set of edges allowed in the input document tree of S.

$N_O^{min}(S)$ *is the set of nodes required in the output document tree of* S.
$N_O^{max}(S)$ *is the set of nodes allowed in the output document tree of* S.
$E_O^{min}(S)$ *is the set of edges required in the output document tree of* S.
$E_O^{max}(S)$ *is the set of edges allowed in the output document tree of* S.

Proof We prove the first four equations. The last four can be derived from the first four and Lemma 3. The first four equations will be proven by showing two inclusions. Using the extensionality axiom we get the equality of the sets. For any document tree T we use the following notation: N_T is the set of nodes of T and E_T is the set of edges of T.

- Suppose that $n \in N_I^{min}(S)$ is not in the input document tree T of S. From the definition of $N_I^{min}(S)$ we know that the first occurence of n is either in a deletion or in an addition as parent node. Since n is not in the document tree at the time of this first operation, the operation (and hence $S[T]$) is undefined.
  Suppose that n in required in the input document tree of S (i.e., $(\forall T).(S[T]$ defined $\rightarrow n \in N_T)$), but not in $N_I^{min}(S)$. From Lemma 2 follows that at least one action on n is needed, hence $\phi_S(n, \text{add}(m, l, n))$ holds (since $n \notin N_I^{min}(S)$). But this action requires that n is not in the document tree at the time of the operation (i.e.. $(\forall T).(S[T]$ defined $\rightarrow n \notin N_T)$). This results in $(\forall T).(S[T]$ defined $\rightarrow n \in N_T \wedge n \notin N_T)$, which is only true if S is incorrect (but S is supposed to be correct).

- Let n be a node that is in the input document tree T of S, but not in the set $N_I^{max}(S)$. Then we know by the definition that n occurs in S and that the first occurence of n in S is as a child node in an addition. Since n would be in the document tree when the edge (m, l, n) is added, the addition (and hence $S[T]$) would be undefined. Therefore n is not allowed in the input document tree of S.
  Suppose that $n \in N_I^{max}(S)$ and that n is not allowed in the input document tree of S (i.e., $(\forall T).(n \in N_T \rightarrow S[T]$ undefined)). If n does not appear in any action of S, n cannot cause S to be undefined (according to Lemma 2). Hence n has to appear in at least one action of S. Since $n \in N_I^{max}(S)$, we know that the first action is not the addition of an edge with n as child node and hence $n \in N_I^{min}(S)$. But from the previous part of this proof we know $n \in N_I^{min}(S) \rightarrow ((\forall T).(S[T]$ defined $\rightarrow n \in N_T))$. Hence $(\forall T).(S[T]$ defined $\rightarrow S[T]$ undefined). This is only true when $S[T]$ is never defined, and hence S is incorrect. But this is a contradiction of our assumption that S is correct.

- Suppose that $(m, l, n) \in E_I^{min}(S)$ is not in the input document tree T of S. From the definition we know that (m, l, n) occurs in S and that the first occurence of this edge in S is its deletion. At the time of this deletion (m, l, n) is not in the document tree, so the operation (and hence $S[T]$) is undefined. Therefore (m, l, n) has to be in the input document tree of S.
  Suppose that $(m, l, n) \notin E_I^{min}(S)$ and (m, l, n) is required in any input document tree T of S (i.e. $(\forall T).(S[T]$ defined $\rightarrow (m, l, n) \in E_T)$). Since (m, l, n) is required, there has to be at least one operation on this edge (this follows from Lemma 2). But we know that the first operation has to be an addition (since $(m, l, n) \notin E_I^{min}(S)$). Hence $(\forall T).(S[T]$ defined $\rightarrow (m, l, n) \notin E_T)$. From this we can conclude that S has to be incorrect, but this is a contradiction with our assumption.

- Let (m, l, n) be an edge that is in the input document tree T of S, but not in $E_I^{max}(S)$. This means that the first action (if any) on (m, l, n) is not a deletion (i.e., $(m, l, n) \notin E_I^{min}(S)$ and that $(m, l, n) \notin \{(m, l, n)|$ no (m_1, l_1, m) nor (m_1, l_1, n) occurs in $S\}$ (i.e., an operation on an edge that ends in m or n occurs in S). If the first occurence of (m, l, n) is an addition then this operation would be undefined, since (m, l, n) is still in the document tree. Therefore (m, l, n) does not occur in an operation of S. We then have two possibilities: (m_1, l_1, n) occurs in S or (m_1, l_1, m) occurs in S. For the edge (m_1, l_1, n) no addition (since we would have two edges ending in n) nor deletion (since this edge cannot be in T) can occur in S. But also the edge (m_1, l_1, m) cannot have an addition (since this would result in two edges ending in m) or a deletion (since the node m still has an outgoing edge) in S. Hence no (m_1, l_1, m) nor (m_1, l_1, n) occurs in S, which is a contradiction with our assumption. Therefore (m, l, n) is not allowed in the input document tree T of S.
  Suppose that $(m, l, n) \in E_I^{max}(S)$. If $(m, l, n) \in E_I^{min}(S)$, then it is easy to see that (m, l, n) is allowed in the input tree, since S is correct and (m, l, n) is required. Therefore suppose that no (m_1, l_1, n) nor (m_1, l_1, m) appears in S. We then have to show that there exists a document tree that contains (m, l, n) and on which the schedule S is defined. Since no (m_1, l_1, m) nor (m_1, l_1, n) appears in S we see that (m, l, n) is allowed in the input document tree. This follows from Lemma 2.

■

We will prove in Theorem 1 that the application of a correct schedule S is defined on each basic-input-tree of S.

Lemma 5 *$N_I^{max}(S)$, $E_I^{max}(S)$, $N_O^{max}(S)$ and $E_O^{max}(S)$ are in general infinite, but they can be represented in a finite way:*

- $N_I^{max}(S)$ *by* $\{n \mid \phi_S(n, \text{add}(m, l, n))\}$
- $E_I^{max}(S)$ *by* $E_I^{min}(S) \cup \{n \mid$ there is a (m_1, l_1, n) that occurs in $S\}$
- $N_O^{max}(S)$ *by* $\{n \mid \lambda_S(n, \text{del}(m, l, n))\}$
- $E_O^{max}(S)$ *by* $E_O^{min}(S) \cup \{n \mid$ there is a (m_1, l_1, n) that occurs in $S\}$.

Proof The representation of $N_I^{max}(S)$ and $N_O^{max}(S)$ is a direct result of the definition, i.e., we represent these sets by their complements in a unique way. In order to prove that the finite representations of $E_I^{max}(S)$ and $E_O^{max}(S)$ are valid, it suffices that we show that the set $A_S = \{n \mid$ there is a (m_1, l_1, n) that occurs in $S\}$ is finite and that it is a valid representation of $B_S = \{(m, l, n) \mid$ no (m_1, l_1, m) nor (m_1, l_1, n) occurs in $S\}$, since we know from the definition that $E_I^{min}(S)$ and $E_O^{min}(S)$ are finite. In order to show this we have to show that $f : \mathcal{A} \to \mathcal{B} : A_S \mapsto \{(m, l, n)|(m \notin A_S) \wedge (n \notin A_S)\}$ is bijection between $\mathcal{A} = \{A_S|S$ is a QL schedule$\}$ and $\mathcal{B} = \{B_S|S$ is a QL schedule$\}$. Obviously f is a (total) function from $\mathcal{A}$ to $\mathcal{B}$. Suppose that $f(A_{S_1}) = f(A_{S_2})$. Then $\{(m, l, n)|(m \notin A_{S_1}) \wedge (n \notin A_{S_1})\} = \{(m, l, n)|(m \notin A_{S_2}) \wedge (n \notin A_{S_2})\}$. Suppose that $A_1 \neq A_2$ and that (without loss of generality) $a \in (A_{S_1} - A_{S_2})$ (i.e., $a \in A_{S_1}$ and $a \notin A_{S_2}$. Then the edges with a as child or parent node are in $f(A_{S_2})$ but not in $f(A_{S_1})$. Hence if $f(A_{S_1}) = f(A_{S_2})$ then $A_1 = A_2$, which means that f is an injection. Furthermore $image(f) = \mathcal{B}$, since if $B_S = \{(m, l, n) \mid$ no (m_1, l_1, m) nor

(m_1, l_1, n) occurs in $S\}$ then there exists an $A_S = \{n \mid$ there is a (m_1, l_1, n) that occurs in $S\}$ such that $B_S = f(A_S)$. Therefore f is a surjection and hence a bijection from $\mathcal{A}$ to $\mathcal{B}$. ∎

Lemma 6 *$N_I^{min}(S), N_I^{max}(S), E_I^{min}(S), E_I^{max}(S), N_O^{min}(S), N_O^{max}(S), E_O^{min}(S)$ and $E_O^{max}(S)$ can be calculated in $O(n_a.log(n_a))$-time ($O(n_S.log(n_S))$-time) and in $O(n_a)$-space ($O(n_S)$-space), n_a being the number of actions in S.* ∎

Proof For the computation of the N and E-sets we need to know which node or edge has which first or last operation in the schedule S. For $E_I^{max}(S)$ and $E_O^{max}(S)$ we also need to know the nodes n for which a (m_1, l_1, n) occurs in S (this follows from Lemma 5). For the first part we will create a list of (node, action)- or (edge, action)-pairs in the same order as the actions appear in S. After this we will do a stable sort on this list on the node or edge, so all (node, action)- or (edge, action)-pairs of the same node or edge are brought together, but still appear in the same relative order (w.r.t. each other) in the list. This can be done in $O(n_a.log(n_a))$ time and $O(n_a)$ space. Now we can look for the first or last occurence of a node or edge in $O(n_a)$ time and depending on this action we can see whether this node or edge will be in the computed set. In order to determine for which nodes n a (m_1, l_1, n) occurs in S we need $O(n_a)$ time. Hence the computation of the N- and E-sets can be done in $O(n_a.log(n_a))$-time ($O(n_S.log(n_S))$-time) and $O(n_a)$-space ($O(n_S)$-space). ∎

Lemma 7 *For any node n and for any edge (m, l, n), it is decidable whether n is in $N_I^{min}(S)$, $N_I^{max}(S)$, $N_O^{min}(S)$, or $N_O^{max}(S)$ and whether (m, l, n) is in $E_I^{min}(S)$, $E_I^{max}(S)$, $E_O^{min}(S)$ or $E_O^{max}(S)$ in $O(n_a)$-time ($O(n_S) - time$) and in $O(log(n_a))$-space ($O(log(n_S))$-space), n_a being the number of actions in S.*

Proof In order to decide whether n is in $N_I^{min}(S)$ or $N_I^{max}(S)$, we need to look for the first occurence of a node in an action. We do this by running over all actions in order of the schedule. If the first action of the node n is an addition with n as parent node, we accept for $N_I^{min}(S)$ and reject for $N_I^{max}(S)$. If no action involving n is encountered, we reject for $N_I^{min}(S)$ and accept for $N_I^{max}(S)$. Anologously we obtain results for $N_O^{min}(S)$ and $N_O^{max}(S)$. It's easy to see that the we need $O(n_a)$-time. The space requirements ($O(log(n_a))$) are for the counter used to run over all actions.
For deciding whether (m, l, n) is in $E_I^{min}(S)$ or $E_I^{max}(S)$ we follow the same strategy. If the first action is the deletion of this edge, then we accept for $E_I^{min}(S)$, else we reject. For deciding $E_I^{max}(S)$, we need two runs over the set of actions. In the first run we accept an edge (m, l, n) if its first occurence is a deletion. If this is not true, then we'll execute a second run over all actions. If (m_1, l_1, m) or (m_1, l_1, n) occurs in S (for any m_1, l_1) then we reject. If no acception nor rejection happened during these two runs, we accept the edge for $E_I^{max}(S)$. Obviously analogous results can be achieved for $E_O^{min}(S)$ and $E_O^{max}(S)$, and the space and time complexity is respectively $O(log(n_a))$ ($O(log(n_S))$) and $O(n_a)$ ($O(n_S)$). ∎

For the N-sets and the E-sets we can derive the following properties:

Property 1 *If S is a correct QL schedule then $E_I^{min}(S)$ and $E_O^{min}(S)$ are forests.*

Proof Since S is a correct QL schedule there exists a document tree T such that $S[T]$ is defined (and hence $S[T]$ is also a tree). We know from Lemma 4 that $E_I^{min}(S)$ and $E_O^{min}(S)$ are subgraphs of respectively T and $S[T]$ (which are trees) and hence are forests. ∎

The following lemma establishes the relationships between the addition and deletion sets, and the basic input and output sets.

Lemma 8 *Let S be a correct QL schedule.*
$E_O^{min}(S) = E_I^{min}(S) - \mathrm{DEL}(S) \cup \mathrm{ADD}(S)$
$E_O^{max}(S) = E_I^{max}(S) - \mathrm{DEL}(S) \cup \mathrm{ADD}(S)$

Proof

- Suppose that (m, l, n) is an edge in $E_O^{min}(S)$ and hence by Lemma 4 required in the output. An edge is required in the output iff all DTs T for which $S[T]$ is defined have this edge in $S[T]$. From Lemma 1 follows that $S[T] = T \cup \mathrm{ADD}(S) - \mathrm{DEL}(S)$ for all basic-input-trees T. Hence if $S[T]$ is defined, then the edge (m, l, n) is required in $T \cup \mathrm{ADD}(S) - \mathrm{DEL}(S)$. This means that $(m, l, n) \notin \mathrm{DEL}(S)$ and either (m, l, n) is in $\mathrm{ADD}(S)$, or (m, l, n) is required in T, but then it is required in the input and hence in $E_I^{min}(S)$.
  Suppose that $(m, l, n) \in E_I^{min}(S) - \mathrm{DEL}(S) \cup \mathrm{ADD}(S)$. Then we know by Lemma 1 that for all DTs S for which $S[T]$ is defined, the edge (m, l, n) has to be in the output of S, since it was required in the input and if an action occured on S, then the last one was the addition of this edge. Hence $(m, l, n) \in E_O^{min}(S)$.

- Suppose that (m, l, n) is an edge in $E_O^{max}(S)$ and hence by Lemma 4 allowed in the output. An edge is allowed in the output iff there is a DT T (say T') for which $S[T]$ is defined that has this edge in $S[T]$. From Lemma 1 follows that $S[T] = T \cup \mathrm{ADD}(S) - \mathrm{DEL}(S)$ for all basic-input-trees T. Hence the edge (m, l, n) is allowed in $T' \cup \mathrm{ADD}(S) - \mathrm{DEL}(S)$. This means that $(m, l, n) \notin \mathrm{DEL}(S)$ and either (m, l, n) is in $\mathrm{ADD}(S)$, or (m, l, n) is allowed in T, but then it is allowed in the input and hence in $E_I^{max}(S)$.
  Suppose that $(m, l, n) \in E_I^{max}(S) - \mathrm{DEL}(S) \cup \mathrm{ADD}(S)$. Then we know by Lemma 1 that there is a DTs T for which $S[T]$ is defined and that has (m, l, n) in the output of S, since it was allowed in the input and if an action occured on S, then the last one was the addition of this edge. Hence $(m, l, n) \in E_O^{max}(S)$.

■

3.3 C-Condition

When a QL schedule is not correct this is always because two operations in the QL schedule interfere, as for example the two operations in the incorrect transaction of Example 4: $(\mathrm{add}(n_1, l_1, n), t_1)$ and $(\mathrm{add}(n_2, l_2, n), t_1)$. If these two operations immediately follow each other then at least one of them will always fail. However, if between them we find the action $\mathrm{del}(n_1, l_1, n)$ then this does no longer hold. The following definition attempts to identify such pairs of interfering operations and states which operations we should find between them to remove the interference.

Definition 7 *A QL schedule fulfills the* C-condition *iff all following subconditions are fulfilled*

1. *If the actions* $(\mathrm{add}(n, l_1, n_1), t_1)$ *and* $(\mathrm{add}(n_2, l_2, n), t_2)$ *appear in that order in S then the action* $(\mathrm{del}(n, l_1, n_1), t_3)$ *appears between them.*

2. *If the actions* $(\text{add}(n_1, l_1, n), t_1)$ *and* $(\text{add}(n_2, l_2, n), t_2)$ *appear in that order in* S *then the action* $(\text{del}(n_1, l_1, n), t_3)$ *appears between them.*

3. *If the actions* $(\text{add}(n, l_1, n_1), t_1)$ *and* $(\text{del}(n_2, l_2, n), t_2)$ *appear in that order in* S *then the action* $(\text{del}(n, l_1, n_1), t_3)$ *appears between them.*

4. *If the actions* $(\text{add}(n_1, l_1, n), t_1)$ *and* $(\text{del}(n, l_2, n_2), t_2)$ *appear in that order in* S *then the action* $(\text{add}(n, l_2, n_2), t_3)$ *appears between them.*

5. *If the actions* $(\text{add}(n_1, l_1, n), t_1)$ *and* $(\text{del}(n_2, l_2, n), t_2)$ *appear in that order in* S *and* $(n_1, l_1) \neq (n_2, l_2)$ *then the action* $(\text{del}(n_1, l_1, n), t_3)$ *appears between them.*

6. *If the actions* $(\text{del}(n, l_1, n_1), t_1)$ *and* $(\text{add}(n_2, l_2, n), t_2)$ *appear in that order in* S *then an action of the form* $(\text{del}(n_3, l_3, n), t_3)$ *appears between them.*

7. *If the actions* $(\text{del}(n_1, l_1, n), t_1)$ *and* $(\text{add}(n, l_2, n_2), t_2)$ *appear in that order in* S *then an action of the form* $(\text{add}(n_3, l_3, n), t_3)$ *appears between them.*

8. *If the actions* $(\text{del}(n_1, l_1, n), t_1)$ *and* $(\text{del}(n, l_2, n_2), t_2)$ *appear in that order in* S *then an action of the form* $(\text{add}(n_3, l_3, n), t_3)$ *appears between them.*

9. *If the actions* $(\text{del}(n_1, l_1, n), t_1)$ *and* $(\text{del}(n_2, l_2, n), t_2)$ *appear in that order in* S *then the action* $(\text{add}(n_2, l_2, n), t_3)$ *appears between them.*

We will use the C-condition for deciding whether a QL schedule is correct. Before we give the theorems (and proofs) for this, we show an example of an incorrect schedule that does not satisfy the C-condition.

Example 5 *The next schedule is incorrect:*
$(\text{add}(r, l_1, n_1), t_1), (\text{add}(n_1, l_2, n_2), t_1), (\text{add}(n_2, l_3, n_3), t_1), (\text{del}(r, l_1, n_1), t_2)$
We see that the C-condition does not hold, since the third subcondition does not hold. One way to fix this problem is by inserting the action $(\text{del}(n_1, l_2, n_2), t_3)$ *right before the first* del *statement. In fact we see the same subcondition will again cause a problem, so we will have to insert* $(\text{del}(n_2, l_3, n_3), t_3)$ *right after the last* add *statement. This causes the schedule to be correct.*

The following theorem establishes the relationship between correctness, basic-input-trees and the C-condition.

Theorem 1 *The following conditions are equivalent for a QL schedules* S*:*

1. *there is a basic-input-tree of* S *and the application of* S *is defined on each basic-input-tree of* S*.*

2. *there is a basic-input-tree of* S *on which the application of* S *is defined;*

3. S *is correct;*

4. S *fulfills the C-condition;*

5. *there is a tree on which the application of* S *is defined and all trees on which the application of* S *is defined are basic-input-trees of* S*.*

Proof We prove this theorem by proving 6 implications. Using these 6 implications we can deduce every condition from another condition, what will conclude our proof.

- **1 → 2.** Since there is a basic-input-tree T of S and the application of S is defined on each basic-input-tree of S, the application of S on the basic-input-tree T of S is defined.

- **2 → 3.** This follows from the definition of correctness of a schedule.

- **3 → 4.** Suppose S is correct and S does not fulfill the C-condition. Then at least one of the nine subconditions of the C-condition is not fulfilled, i.e., o_3 does not appear between o_1 and o_2 (this is the general form of all subconditions). But in this case the operation o_2 is not defined on the document tree at the time of the execution of o_2 since either an edge (or a node) is present in the document tree right before o_2 and this edge (or node) is not allowed or an edge (or a node) is not present in the document tree right before o_2 and this edge (or node) is needed. Hence S is not correct. For instance suppose that the seventh subcondition of the C-condition does not hold. Then there is never an edge ending in n in the schedule at the time of the addition of (n, l_2, n_2) and hence (since n is not the root) the addition of this edge is always undefined, so S is not correct.

- **4 → 1.** First we prove that there is a basic-input-tree of S. Therefore we show that $E_I^{min}(S)$ is a forest if S fulfills the C-condition and use this forest to construct a tree. Suppose $E_I^{min}(S)$ is not a forest and S fulfills the C-condition. Then either two edges of $E_I^{min}(S)$ end in the same node or $E_I^{min}(S)$ contains a cycle. If $E_I^{min}(S)$ has two edges (m_1, l_1, n) and (m_2, l_2, n) ending in the same node n then for both (m_1, l_1, n) and (m_2, l_2, n) the first operation in S is its deletion. Suppose (without loss of generality) that (m_1, l_1, n) is deleted before (m_2, l_2, n). From the ninth subcondition of the C-condition then follows that add(m_2, l_2, n) has to appear between the two deletions in S. But then $(m_2, l_2, n) \notin E_I^{min}(S)$. Therefore suppose $E_I^{min}(S)$ contains a cycle $(n_1, l_1, n_2), (n_2, l_2, n_3), ..., (n_k, l_k, n_1)$. Then the first action for all these edges is the deletion. Suppose that the first deletion of (n_1, l_1, n_2) occurs before the first deletion of (n_2, l_2, n_3). Then from the eighth and fourth subcondition of the C-condition follows that add(n_2, l_2, n_3) has to occur between them. But then the first action on (n_2, l_2, n_3) is the addition and hence this edge is not in $E_I^{min}(S)$. From this follows that the first occurence del(n_2, l_2, n_3) has to appear before the first occurence of del(n_1, l_1, n_2). For all other edges that share a node we find a similar result. Therefore the first occurence of del(n_1, l_1, n_2) has to occur before the first occurence of del(n_k, l_k, n_1), which has to occur (omitting some intermediate steps) before (n_2, l_2, n_3). This is a contradiction, hence $E_I^{min}(S)$ is a forest.

  Let r be a node that does not occur in an action of S and let T be the graph $E_I^{min}(S)$ augmented with r and all nodes of $N_I^{min}(S)$. Add to the graph T edges from r to the roots of T. This (new) T is a basic-input-tree since T is a tree and the new edges of T are also in $E_I^{max}(S)$. This follows from the fact that no action on an edge ending in r (since r is chosen not to appear in S) nor action on an edge ending in a root r' of $E_I^{min}(S)$ appears in S. Indeed, suppose that an action on an edge ending in a root r' of $E_I^{min}(S)$ appears in S. If such an action appears in S, then the first action on this edge is of the form add(m_1, l_1, r') (since in case of a deletion r' would not be a root of $E_I^{min}(S)$). If the first addition add(m_1, l_1, r') occurs before the first

deletion of an edge $(r', l_2, n_2) \in E_I^{min}(S)$ then it follows from the fourth subcondition of the C-condition that the addition of the edge (r', l_2, n_2) appears in between. Hence $(r', l_2, n_2) \notin E_I^{min}(S)$. If the first addition $\text{add}(m_1, l_1, r')$ occurs after the first deletion of an edge $(r', l_2, n_2) \in E_I^{min}(S)$ then it follows from the sixth subcondition of the C-condition that an action of the form $\text{del}(n_3, l_3, r')$ has to occur in between, but then r' is not a root of $E_I^{min}(S')$.

We now prove that the application of S is defined on each basic-input-tree of S. We do this by induction on the length of S. First let S be the empty schedule. It is easy to see that the application of S is defined on each document tree and hence on each basic-input-tree of S. Suppose that the application of S of length $n_a < K$ is defined on each basic-input-tree (induction hypothesis). A schedule S of length K can be written as the schedule $o.S'$, where S' is a schedule of length $K-1$ and o is the first operation of S. From the induction hypothesis follows that S' is defined on each basic-input-tree of S'. We now have to show that o is defined on each basic-input-tree of S and that for each basic-input-tree T of S the application $o[T]$ is a basic-input-tree of S'. The operation o is one of the two following operations:

- $\text{add}(m, l, n)$: Suppose that $o[T]$ is not defined. This is only possible if an edge of the form $(m_1, l_1, n) \in E_T \subseteq E_I^{max}(S)$ exists (since the first action on (m, l, n) is the addition, it follows that $(m_1, l_1) \neq (m, l)$) or $m \notin N_T \supseteq N_I^{min}(S)$. The first case ($(m_1, l_1, n) \in E_I^{max}(S)$) is impossible since an action occurs on n in S ($\text{add}(m, l, n)$) and $(m_1, l_1, n) \notin E_I^{min}(S)$. Indeed, if $(m_1, l_1, n) \in E_I^{min}(S)$ then the first action on (m_1, l_1, n) is a deletion and the operations $\text{add}(m, l, n)$ and $\text{del}(m_1, l_1, n)$ occur in that order, but then the fifth and ninth subcondition of the C-condition require the addition of (m_1, l_1, n) to appear before $\text{del}(m_1, l_1, n)$ (which is the first action in S on (m_1, l_1, n)). The second case ($m \notin N_I^{min}(S)$) is also impossible since the first action on m in S is $\text{add}(m, l, n)$. Hence $o[T] = T'$ is defined and its result is the following:
  $E_{T'} = E_T \cup \{(m, l, n)\}$
  $N_{T'} = N_T \cup \{n\}$
  Furthermore $E_I^{min}(S) = E_I^{min}(S')$. Suppose T' is not a basic-input-tree of S'. This means that at least one of the following four statements does not hold:
  - $E_I^{min}(S') \subseteq E_{T'}$. Since T is a basic-input-tree of S we know that $E_I^{min}(S) \subseteq E_T$. Hence $E_I^{min}(S') = E_I^{min}(S) \subseteq E_T \subseteq E_{T'}$.
  - $E_{T'} \subseteq E_I^{max}(S')$. Since T is a basic-input-tree of S we know that $E_T \subseteq E_I^{max}(S)$. Furthermore we know that $E_I^{min}(S) = E_I^{min}(S')$ and that $\{(m_1, l_1, n_1) \mid \text{no action on an edge ending in } m_1 \text{ or } n_1 \text{ occurs in } S\} \subseteq \{(m_1, l_1, n_1) \mid \text{no action on an edge ending in } m_1 \text{ or } n_1 \text{ occurs in } S'\}$ (since all actions in S' are also actions in S). Hence $E_I^{max}(S) \subseteq E_I^{max}(S')$. Hence $E_T \subseteq E_I^{max}(S) \subseteq E_I^{max}(S')$. We now have to prove that $(m, l, n) \in E_I^{max}(S')$, which will result in $E_T \cup \{(m, l, n)\} = E_{T'} \subseteq E_I^{max}(S')$. If no action on an edge ending in m or n occurs in S' then clearly $\{(m, l, n)\} \in E_I^{max}(S')$. Else we have four possibilities for the *first* operation on m or n in S', i.e., an edge ending in m or n is deleted or added. Since $S = o.S'$ and S fulfills the C-condition, we know by the first, second, third and fifth subcondition of the C-condition that a deletion of (m, l, n) has to appear in S between o and the first action on

an edge ending in m or n. But this deletion would cause (m, l, n) to be in $E_I^{min}(S') \subseteq E_I^{max}(S')$.

* $N_I^{min}(S') \subseteq N_{T'}$. If this is not true then there is a node $n_1 \in N_I^{min}(S')$ which is not in $N_{T'}$. We know that $n_1 \neq m, n$ since $m, n \in N_{T'}$. But then the first action on n_1 in S' is the same action as the first action on n_1 in S. Hence if $n_1 \in N_I^{min}(S')$ then $n_1 \in N_I^{min}(S) \subseteq N_T \subseteq N_{T'}$.
* $N_{T'} \subseteq N_I^{max}(S')$. If this is not true. then there is a node $n_1 \in N_{T'}$ which is not in $N_I^{max}(S')$. If $n_1 = n$ then the first occurence of n in S' is its addition since $n \notin N_I^{max}(S')$. From the second subcondition of the C-condition then follows that the deletion of n has to occur between o and the first action on n in S'. Else if $n_1 = m$ then it follows from the first subcondition of the C-condition that the addition of m has to occur between o and the first action on m in S'. Both cases result in a contradiction since $S = o.S'$. Therefore $n_1 \neq m, n$, so the first occurence of n_1 is the same action in S as it is in S'. But since $n_1 \in (N_{T'} - \{n\}) = N_T$ we know that the first occurence of n_1 in S and hence S' is not its addition. Hence $n_1 \in N_I^{max}(S')$, which is a contradiction.

Since all four statements hold it follows that if T is a basic-input-tree of $S = o.S'$ then $o[T]$ is defined and is a basic-input-tree of S'.

– del(m, l, n): Suppose that $o[T]$ is not defined. This is only possible if $(m, l, n) \notin E_T \supseteq E_I^{min}(S)$ or an edge of the form $(n, l_1, n_1) \in E_T \subseteq E_I^{max}(S)$. The first case (i.e., $(m, l, n) \notin E_I^{min}(S)$) is impossible since the first operation on (m, l, n) is the deletion and hence (m, l, n) is by definition in $E_I^{min}(S)$. The second case (i.e., $(n, l_1, n_1) \in E_I^{max}(S)$) is also impossible since an action occurs on n in S (del(m, l, n)) and $(n, l_1, n_1) \notin E_I^{min}(S)$. Indeed, if $(n, l_1, n_1) \in E_I^{min}(S)$ then the first operation on (n, l_1, n_1) is its deletion. Hence del(m, l, n) and del(n, l_1, n_1) appear in that order in S without any action on (n, l_1, n_1) between them. But this contradicts the eighth and fourth subcondition of the C-condition, which says that add(n, l_1, n_1) has to appear between them. Hence $o[T] = T'$ is defined and its result is the following:
$E_{T'} = E_T - \{(m, l, n)\}$
$N_{T'} = N_T - \{n\}$
Furthermore $E_I^{min}(S') = E_I^{min}(S) - \{(m, l, n)\}$, since for all edges the first action on it in S is the same as in S' except for (m, l, n). However if $(m, l, n) \in E_I^{min}(S')$ then its first action in S' is its deletion, but then the ninth subcondition of the C-condition will require to have the addition between o and the first action on this edge in S', which is a deletion, and hence the addition is the first action on (m, l, n) in S'. Suppose T' is not a basic-input-tree of S'. This means that at least one of the following four statements has to hold:

* $E_I^{min}(S') \subseteq E_{T'}$. Since T is a basic-input-tree of S we know that $E_I^{min}(S) \subseteq E_T$ and hence $E_I^{min}(S') = E_I^{min}(S) - \{(m, l, n)\} \subseteq E_T - \{(m, l, n)\} = E_{T'}$
* $E_{T'} \subseteq E_I^{max}(S')$. Since T is a basic-input-tree of S we know that $E_T \subseteq E_I^{max}(S)$ and hence $E_{T'} = E_T - \{(m, l, n)\} \subseteq E_I^{max}(S) - \{(m, l, n)\}$. We still have to prove $E_I^{max}(S) - \{(m, l, n)\} \subseteq E_I^{max}(S')$. This follows from $E_I^{min}(S') = E_I^{min}(S) - \{(m, l, n)\}$ and $\{(m_1, l_1, n_1) \mid$ no action on an edge ending in m_1 or n_1 occurs in $S\} \subseteq \{(m_1, l_1, n_1) |$ no action on an edge ending in m_1 or n_1 occurs in $S'\}$ (since all actions in S' are also actions in S).

* $N_I^{min}(S') \subseteq N_{T'}$. If this is not true then there is a node $n_1 \in N_I^{min}(S')$ which is not in $N_{T'}$. If $n_1 = n$ then the seventh, eighth and ninth subcondition of the C-condition say that between $\text{del}(m, l, n)$ and the first operation that causes n_1 to be in $N_I^{min}(S')$ an addition of n_1 has to appear. But since $S = o.S'$ this addition has to be in S' and hence $n_1 \notin N_I^{min}(S')$. If $n_1 = m$ then $n_1 \in N_{T'}$. Therefore suppose $n_1 \neq m, n$. Then the first occurence of n_1 in an action is the same in S' as it is in S. Hence if $n_1 \in N_I^{min}(S')$ then $n_1 \in N_I^{min}(S) \subseteq N_T = (N_{T'} - \{n\})$ and hence $n_1 \in N_{T'}$.
* $N_{T'} \subseteq N_I^{max}(S')$. If this is not true then there is a node $n_1 \in N_{T'}$ which is not in $N_I^{max}(S')$. Since $n_1 \in N_{T'}$ we know that $n_1 \neq n$. If $n_1 = m$ then $\text{del}(m, l, n)$ and $\text{add}(m_1, l_1, m)$ has to appear in that order, where $\text{add}(m_1, l_1, m)$ is the first action on m in S'. From the sixth subcondition of the C-condition then follows that an action of the form $\text{del}(m_2, l_2, m)$ has to appear between these two actions, but then the addition of m is not the first occurence of m in S'. Hence $n_1 \neq m, n$, so the first occurence of n_1 in S' is the same as it is in S. Since we also know that $n_1 \in (N_{T'} - \{n\} = N_T \subseteq N_I^{max}(S)$ it follows that $n \in N_I^{max}(S')$.

Since all four statements hold it follows that if T is a basic-input-tree of $S = o.S'$ then $o[T]$ is defined and is a basic-input-tree of S'.

- **3 → 5.** From Lemma 4 and Definition 5 follows that any input document tree T for which $S[T]$ is defined has to be a basic-input-tree. Furthermore there has to be at least one such T, since S is correct.

- **5 → 3.** Since there is at least one tree on which the application of S is defined and since this tree has to be a basic-input-tree, there is at least one basic-input-tree on which the application of S is defined and hence S is correct by definition.

■

Corollary 1 *It is decidable whether a QL schedule or a transaction is correct in $O(n_a^3)$-time ($O(n_S^3)$-time) and $O(log(n_a))$-space ($O(log(n_S))$-space), n_a being the number of actions.*

Proof We first show that it is decidable whether a QL Shedule S fulfills the C-condition in $O(n_a^3)$-time and $O(log(n_a))$-space.

All nine subconditions of the C-condition involve checking whether a given action appears between two other (interfering) actions. This can be done by the following algorithm:

```
1 proc C-condition(S)
2   interferenceFound :=false
3   for i := 0 to n_a
4       for j := i + 2 to n_a
5           if possible interference
6             then
7                   OK :=false
8                   for k := i + 1 to j − 1
```

```
9                       if action_k is needed between action_i and action_j to resolve the interference
10                        then OK :=true
11                      fi
12                  end
13                  if ¬OK
14                    then interferenceFound :=true
16                  fi
17          fi
18      end
19  end
20  C-condition := ¬interferenceFound
```

The outer loop (lines (3)-(19)) will be executed at most n_a times. The middle loop (lines (4)-(18)) will be executed at most $n_a - 2$ times each execution of the outer loop. In each middle loop we will have at most $n_a - 2$ iterations each time of the inner loop (lines (8)-(12)). This leads us to an upperbound of $O(n_a(n_a - 2)^2) = O(n_a^3)$ for the time needed to decide the C-condition. Furthermore only three pointers to nodes are needed in this algorithm. We assume that the number of nodes is dependent of the number of actions, so each pointer needs $O(log(n_a))$ space.

We just showed that we can decide whether a QL schedule S fulfills the C-condition in $O(n_a^3)$-time and $O(log(n_a))$-space. Hence it is decidable, by consequence of Theorem 1, whether a QL schedule S is correct in $O(n_a^3)$-time and $O(log(n_a))$-space ($O(n_S^3)$-time and $O(log(n_S))$-space). ■

Chapter 4

Equivalence and Serializability of QL Schedules

The purpose of a scheduler is to interleave requests by processes such that the resulting schedule is serializable. This can be done by deciding for each request whether the schedule extended with the requested operation is still serializable, without looking at the instance. In this section we discuss the problem of deciding whether a correct QL schedule is serializable and whether two correct QL schedules are equivalent.
One possible approach for a scheduler could be to introduce a locking mechanism such that operations of a certain process would only be allowed if they do not require locks that conflict with locks required by earlier operations. Because operations with non-conflicting locks can be commuted any schedule that is allowed by such a scheduler can be serialized. The following example shows however that the reverse does not hold: Indeed, the next QL schedule

$$S = (\text{add}(r, l_1, n_1), t_1),\ (\text{del}(r, l_1, n_1), t_2),\ (\text{add}(r, l_2, n_2), t_2),\\ (\text{del}(r, l_2, n_2), t_2),\ (\text{add}(r, l_2, n_2), t_1),\ (\text{del}(r, l_2, n_2), t_1).$$

is correct since it is defined on $T = (\{r\}, \emptyset, r)$. Furthermore it is serializable, and the equivalent serial QL schedules are

$$S_1 = (\text{add}(r, l_1, n_1), t_1),\ (\text{add}(r, l_2, n_2), t_1),\ (\text{del}(r, l_2, n_2), t_1),\\ (\text{del}(r, l_1, n_1), t_2),(\text{add}(r, l_2, n_2), t_2),\ (\text{del}(r, l_2, n_2), t_2)$$

and

$$S_2 = (\text{del}(r, l_1, n_1), t_2),(\text{add}(r, l_2, n_2), t_2),\ (\text{del}(r, l_2, n_2), t_2)\\ (\text{add}(r, l_1, n_1), t_1),\ (\text{add}(r, l_2, n_2), t_1),\ (\text{del}(r, l_2, n_2), t_1).$$

but we cannot go from S to S_1 or to S_2 only by swapping such that the intermediate schedules are correct.
Therefore we will investigate in the following sections the possibility of an algorithm that *exactly* decides serializability.

4.1 Deciding Equivalence

A subproblem of deciding serializability is deciding equivalence. It can be shown that the application of two QL schedules over the same set of transactions on the same DT T result in the same DT, if they are both defined.

Lemma 9 *Let S and S' be two QL schedules over the same set of transactions. $S[T] = S'[T]$ if $S[T]$ and $S'[T]$ are both defined.*

Proof Suppose (m, l, n) is an arbitrary edge that appears in S (and hence S'). Since S and S' are two correct QL schedules over the same set of transactions, the number of deletions and additions of (m, l, n) are equal for both schedules and (m, l, n) is alternatively added and deleted. Suppose (without loss of generality) that the first operation on (m, l, n) is the addition in S and the deletion in S'. Since S and S' are both defined on T, we get a contradiction. Therefore the first operation on the edge (m, l, n) is the same on both schedules and hence the last operation on this edge is for both schedules the same. Since this holds for any edge (m, l, n) that occurs in S and S', we know that $\text{ADD}(S) = \text{ADD}(S')$ and $\text{DEL}(S) = \text{DEL}(S')$. Hence we conclude (using Lemma 1) that $S[T] = T \cup \text{ADD}(S) - \text{DEL}(S) = T \cup \text{ADD}(S') - \text{DEL}(S') = S'[T]$. ■

As a consequence the problem of deciding whether two correct schedules over two given transactions are equivalent reduces to the problem of deciding whether their result is defined for the same DTs, which in turn can be decided with the help of the basic input and output sets.

Theorem 2 *Two correct QL schedules S_1, S_2 over the same set of transactions are equivalent iff they have the same set of basic-input-trees, i.e., iff $N_I^{min}(S_1) = N_I^{min}(S_2)$, $N_I^{max}(S_1) = N_I^{max}(S_2)$, $E_I^{min}(S_1) = E_I^{min}(S_2)$ and $E_I^{max}(S_1) = E_I^{max}(S_2)$. Hence their equivalence is decidable in $O(n_a.log(n_a))$-time ($O(n_S.log(n_S))$-time) and $O(n_a)$-space ($O(n_S)$-space).*

Proof If two correct schedules have the same set of basic-input-trees then their result is defined for the same set of DTs and hence it follows from Lemma 9 that they are equivalent. Suppose that two correct QL schedules S_1 and S_2 (over the same set of transactions) are equivalent. This means that when they are applied on the same document tree T, they either are both undefined or they are both defined and their result is the same (i.e., $S_1[T] = S_2[T]$). Suppose (without loss of generality) that a node or an edge is required in all document trees T for S_1 and that it is not in all document trees T for which $S_2[T]$ is defined. Then there exists a DT T' so that this node or edge is in T' and $S_1[T']$ is defined and $S_2[T']$ is not defined. This is a contradiction, hence the minimal input sets have to be equal, i.e., $N_I^{min}(S_1) = N_I^{min}(S_2)$, and $E_I^{min}(S_1) = E_I^{min}(S_2)$. Suppose (again, without loss of generality) that a node or an edge is allowed in an input tree (say T') for S_1 and not in any input tree for S_2. Then $S_1[T']$ is defined and $S_2[T']$ is not defined. This is again a contradiction and hence the maximul input sets have to be equal, i.e., $N_I^{max}(S_1) = N_I^{max}(S_2)$, and $E_I^{max}(S_1) = E_I^{max}(S_2)$.

From Lemma 6 follows that the N- and E-sets can be calculated in $O(n_a.log(n_a))$-time ($O(n_S.log(n_S))$-time) and $O(n_a)$-space ($O(n_S)$-space). In order to decide equivalence we need to calculate the sets for both schedules and compare them to each other (i.e., do some kind of sorting and stepwise compare each element from the first list to the element at the same position in the second list). ■

Note that this theorem does not hold for two arbitrary QL schedules. Indeed $S_1 = (\text{add}(m, l, n), t)$ and $S_2 = (\text{add}(m, l, n), t), (\text{del}(m, l, n), t)$ have the same basic-input-trees and are not equivalent.

4.2 Composing Correct Schedules

We can use the basic input and output sets to decide whether one correct schedule can directly follow another correct schedule without resulting in an incorrect schedule.

Lemma 10 *Let S_1 and S_2 be two correct QL schedules. Let n_a be the number of actions in $S_1.S_2$. $S_1.S_2$ is correct iff $N_I^{min}(S_2) \subseteq N_O^{max}(S_1)$, $E_I^{min}(S_2) \subseteq E_O^{max}(S_1)$, $N_O^{min}(S_1) \subseteq N_I^{max}(S_2)$, $E_O^{min}(S_1) \subseteq E_I^{max}(S_2)$. The correctness of $S_1.S_2$ is decidable in $O(n_a.log(n_a))$-time ($O(n_S.log(n_S))$-time) and $O(n_a)$-space ($O(n_S)$-space).*

Proof We will prove this lemma in three parts. In the first two parts we prove both directions of the *iff* clause (using Lemma 4), in the third part we will show the space and time complexity.

- Suppose $N_I^{min}(S_2) \subseteq N_O^{max}(S_1)$, $E_I^{min}(S_2) \subseteq E_O^{max}(S_1)$, $N_O^{min}(S_1) \subseteq N_I^{max}(S_2)$ and $E_O^{min}(S_1) \subseteq E_I^{max}(S_2)$. Suppose $S_1.S_2$ is not correct. If T is an arbitrary basic-input-tree of S_1 (since S_1 is correct, there exists such a basic-input-tree), then $T' = S_1[T]$ is a basic-output-tree of S_1. Since we assumed that $S_1.S_2$ is not correct, we know that $S_2[T']$ is never defined, for any $T' = S_1[T]$. Hence for every T' there is a node or edge in T' that is not allowed or a node or edge that is needed and that is not in T'. If there is a node (or edge) in T' that is not allowed in an input-tree of S_2, then this node (or edge) is not in $N_I^{max}(S_2)$ (or $E_I^{max}(S_2)$) and hence not in $N_O^{min}(S_1)$ (or $E_O^{min}(S_1)$), so this node (or edge) is not required in T', hence we may choose another T such that $T' = S_1[T]$ does not contain this node (or edge). If there is a node (or edge) that is required and that is not in T' then this node (or edge) is in $N_I^{min}(S_2)$ (or $E_I^{min}(S_2)$) and hence in $N_O^{max}(S_1)$ (or $E_O^{max}(S_1)$) but not in T'. Since $T' = S_1[T]$ for an arbitrary basic-input-tree of S_1, this means that the node (or edge) is in $N_O^{max}(S_1)$ (or $E_O^{max}(S_1)$), but never in the result of S_1, i.e., they are not allowed. This is a contradiction, hence $S_1.S_2$ is correct.

- Suppose $S_1.S_2$ is correct and that at least one of the four inclusions does not hold. We will show that this is impossible, since then the C-condition (see Definition 7) is violated and hence $S_1.S_2$ is not correct (this follows from Theorem 1), which is a contradiction.

  - If $N_I^{min}(S_2) \not\subseteq N_O^{max}(S_1)$ then there is a node $n \in N_I^{min}(S_2)$ such that $n \notin N_O^{max}(S_1)$. Hence the last action on n in S_1 is the deletion of an edge with child node n and the first action on n in S_2 is either an addition with n as parent node or a deletion. But then respectively the seventh, eighth and ninth subcondition of the C-condition are violated.
  - If $E_I^{min}(S_2) \not\subseteq E_O^{max}(S_1)$ then there is an edge $(m, l, n) \in E_I^{min}(S_2)$ such that $(m, l, n) \notin E_O^{max}(S_1)$. Hence the first action on (m, l, n) in S_2 is its deletion, if there is an action on (m, l, n) in S_1 then the last action on this edge in S_1 is not the addition and either (m_1, l_1, n) or (m_1, l_1, m) occur in S_1. If the last occurence of an edge of the form (m_1, l_1, n) is a deletion then the ninth subcondition of the C-condition is violated, else if it is an addition then the fifth subcondition is violated. Hence (m_1, l_1, n) does not occur in S_1, so an edge of the form (m_1, l_1, m) has to occur in S_1. But then the last occurence of this edge makes $S_1.S_2$ dissatisfy the C-condition, since if it is an addition or a deletion then respectively the fourth and eighth subcondition of the C-condition are violated.

– If $N_O^{min}(S_1) \not\subseteq N_I^{max}(S_2)$ then there is a node $n \in N_O^{min}(S_1)$ such that $n \notin N_I^{max}(S_2)$. Hence the last action on n in S_1 is either an addition or a deletion with n as parent node and the first action on n in S_2 is the addition with n as child node. But then respectively the first, second and sixth subcondition of the C-condition are violated.

– If $E_O^{min}(S_1) \not\subseteq E_I^{max}(S_2)$ then there is an edge $(m, l, n) \in E_O^{min}(S_1)$ such that $(m, l, n) \notin E_I^{max}(S_2)$. Hence the last action on (m, l, n) in S_1 is its addition, if (m, l, n) occurs in S_2 then its first occurence is not the deletion and (m_1, l_1, m) or (m_1, l_1, n) occurs in S_2. If (m_1, l_1, n) occurs in S_2 then its first occurence is either an addition or a deletion, but then respectively the second and fifth subconditions of the C-condition are violated. Therefore (m_1, l_1, m) has to occur in S_2, but then again its first occurence is either an addition or a deletion, which respectively violates the first and third subcondition of the C-condition. Hence the C-condition cannot be satisfied.

Hence all four inclusions have to hold, since the C-condition has to be satisfied.

- In order to decide the correctness of $S_1.S_2$ we compute the input-sets of S_2 and the output-sets of S_1 which can be done in $O(n_a.log(n_a))$-time and $O(n_a)$-space (this follows from Lemma 6). We may assume that the results of this computation is ordered, since we have enough time and space to do so. We now have to decide whether one set of length $O(n_a)$ is a subset of another set of length $O(n_a)$, which can be done in $O(n_a)$-time and $O(log(n_a))$ space. Hence we need $O(n_a.log(n_a))$-time ($O(n_S.log(n_S))$-time) and $O(n_a)$-space ($O(n_S)$-space) to decide the correctness of $S_1.S_2$.

■

The following theorems show how the basic input and output sets can be computed for a concatenation of schedules if we know these sets for the concatenated schedules.

Lemma 11 *Let S_1, S_2 and $S_1.S_2$ be three correct QL schedules. Then*

$$\begin{aligned}
N_I^{min}(S_1.S_2) &= N_I^{min}(S_1) \cup (N_I^{max}(S_1) \cap N_I^{min}(S_2)) \\
N_I^{max}(S_1.S_2) &= N_I^{max}(S_1) \cap (N_I^{min}(S_1) \cup N_I^{max}(S_2)) \\
E_I^{min}(S_1.S_2) &= E_I^{min}(S_1) \cup (E_I^{max}(S_1) \cap E_I^{min}(S_2)) \\
E_I^{max}(S_1.S_2) &= E_I^{max}(S_1) \cap (E_I^{min}(S_1) \cup E_I^{max}(S_2))
\end{aligned}$$

Proof We prove this theorem by using Lemma 10 and Definition 5. The equality of the sets will be proven by using the extensionality axiom.

- If a node n is in $N_I^{min}(S_1.S_2)$ then it occurs in an action of $S_1.S_2$ and its first occurence is in a deletion or as a parent node in an addition. If this first occurence is in S_1 then $n \in N_I^{min}(S_1)$. Else $n \in N_I^{min}(S_2)$ and $n \in N_I^{max}(S_1)$, since no operation on n occurs in S_1. Hence if $n \in N_I^{min}(S_1.S_2)$ then $n \in N_I^{min}(S_1) \cup (N_I^{max}(S_1) \cap N_I^{min}(S_2))$.
  If $n \in N_I^{min}(S_1)$ then it occurs in an action of S_1 and its first occurence in S_1 (and hence $S_1.S_2$) is in a deletion or as a parent node in an addition, so $(m, l, n) \in N_I^{min}(S_1.S_2)$. Else if $n \in (N_I^{max}(S_1) \cap N_I^{min}(S_2))$ then we assume that n does not appear in an action of S_1 since otherwise $n \in N_I^{min}(S_1)$. In this case the first occurence in $S_1.S_2$ of n will be in S_2 and since this is in a deletion or as a parent node in an addition, we know that $n \in N_I^{min}(S_1.S_2)$.

- If a node n is in $N_I^{max}(S_1.S_2)$ then its first occurence is not as a child node in an addition iff it occurs in an action of $S_1.S_2$. If it does not occur in $S_1.S_2$ then it does not occur in S_1 nor S_2, hence $n \in N_I^{max}(S_1) \cap N_I^{max}(S_2)$. If it occurs in $S_1.S_2$ then either its first occurence is in S_1, resulting in $n \in (N_I^{max}(S_1) \cap N_I^{min}(S_1))$ (since its first occurence is not as a child node in an addition), or its first occurence is in S_2, resulting in $n \in (N_I^{max}(S_1) \cap N_I^{max}(S_2))$. Hence we see that if $n \in N_I^{max}(S_1.S_2)$ then $n \in (N_I^{max}(S_1) \cap (N_I^{min}(S_1) \cup N_I^{max}(S_2)))$.
  If $n \in N_I^{max}(S_1) \cap N_I^{min}(S_1)$ then n occurs in S_1 and its first occurence in S_1 (and hence in $S_1.S_2$) is not as a child node in an addition, hence $n \in N_I^{max}(S_1.S_2)$. Else if $n \in N_I^{max}(S_1) \cap N_I^{max}(S_2)$ then we may assume that n does not occur in S_1 since otherwise $n \in N_I^{max}(S_1) \cap N_I^{min}(S_1)$. Therefore if n occurs in S_2 and hence in $S_1.S_2$ then its first occurence is not as a child node in an addition, so $n \in N_I^{max}(S_1.S_2)$.

- If an edge (m, l, n) is in $E_I^{min}(S_1.S_2)$ then either the first deletion of the edge (m, l, n) in $S_1.S_2$ is in S_1 or it is in S_2. If it is in S_1 then $(m, l, n) \in E_I^{min}(S_1)$. Else if it is in S_2 then $(m, l, n) \in E_I^{min}(S_2)$. In this case we still have to prove that (m, l, n) is also in $E_I^{max}(S_1)$, since (m, l, n) has to be in $E_I^{max}(S_1) \cap E_I^{min}(S_2)$ (because it cannot be an element of $E_I^{min}(S_1)$). We know from Lemma 10 that $E_I^{min}(S_2) \subseteq E_O^{max}(S_1)$, hence $(m, l, n) \in E_O^{max}(S_1)$. This means that $(m, l, n) \in \{(m, l, n)|$ no (m_1, l_1, m) nor (m_1, l_1, n) occurs in $S\}$, since $(m, l, n) \notin E_O^{min}(S_1)$ (because no operation and hence no addition on (m, l, n) occurs in S_1). Hence $(m, l, n) \in E_I^{max}(S_1)$.
  If $(m, l, n) \in E_I^{min}(S_1)$ then the first occurence in S_1 and hence in $S_1.S_2$ is the deletion of this edge. Hence $(m, l, n) \in E_I^{min}(S_1.S_2)$. Else if $(m, l, n) \in E_I^{max}(S_1) \cap E_I^{min}(S_2)$ then the edge (m, l, n) does not occur in S_1 and the first action on this edge in S_2 (and hence in $S_1.S_2$) is its deletion. Therefore $(m, l, n) \in E_I^{min}(S_1.S_2)$.

- If an edge (m, l, n) is in $E_I^{max}(S_1.S_2)$ then $(m, l, n) \in E_I^{min}(S_1.S_2)$ or $(m, l, n) \in \{(m, l, n)|$ no (m_1, l_1, m) nor (m_1, l_1, n) occurs in $S_1.S_2\}$. Suppose $(m, l, n) \in E_I^{min}(S_1.S_2)$. In this case the first action in $S_1.S_2$ on the edge (m, l, n) is its deletion. If this first deletion is in S_1 then $(m, l, n) \in E_I^{min}(S_1)$ and hence $(m, l, n) \in E_I^{max}(S_1) \cap (E_I^{min}(S_1) \cup E_I^{max}(S_2))$. Else the first action on (m, l, n) in $S_1.S_2$ is the deletion of this edge and occurs in S_2. Hence $(m, l, n) \in E_I^{min}(S_2)$ (and also $(m, l, n) \in E_I^{max}(S_2)$). By Lemma 10 follows that $(m, l, n) \in E_O^{max}(S_1)$. But since (m, l, n) does not occur in S_1, we know that $(m, l, n) \notin E_I^{min}(S_1)$ and hence $(m, l, n) \in \{(m, l, n)|$ no (m_1, l_1, m) nor (m_1, l_1, n) occurs in $S_1\}$. From this follows that $(m, l, n) \in E_I^{max}(S_1)$, hence $(m, l, n) \in E_I^{max}(S_1) \cap (E_I^{min}(S_1) \cup E_I^{max}(S_2))$. Suppose $(m, l, n) \in \{(m, l, n)|$ no (m_1, l_1, m) nor (m_1, l_1, n) occurs in $S_1.S_2\}$. Then we know also that $(m, l, n) \in \{(m, l, n)|$ no (m_1, l_1, m) nor (m_1, l_1, n) occurs in $S_1\}$ and $(m, l, n) \in \{(m, l, n)|$ no (m_1, l_1, m) nor (m_1, l_1, n) occurs in $S_2\}$. Hence $(m, l, n) \in E_I^{max}(S_1)$ and $(m, l, n) \in E_I^{max}(S_2)$, so $(m, l, n) \in E_I^{max}(S_1) \cap (E_I^{min}(S_1) \cup E_I^{max}(S_2))$.
  If $(m, l, n) \in E_I^{max}(S_1) \cap E_I^{min}(S_1)$ then the first occurence in S_1 and hence in $S_1.S_2$ is the deletion of this edge. Hence $(m, l, n) \in E_I^{min}(S_1.S_2) \subseteq E_I^{max}(S_1.S_2)$. Else if $(m, l, n) \in E_I^{max}(S_1) \cap E_I^{max}(S_2)$ then either $(m, l, n) \in E_I^{min}(S_2)$ or $(m, l, n) \in \{(m, l, n)|$ no (m_1, l_1, m) nor (m_1, l_1, n) occurs in $S_2\}$. In the first case we can use the previous part of this lemma to deduce that $(m, l, n) \in E_I^{min}(S_1.S_2) \subseteq E_I^{max}(S_1.S_2)$. In the second case we assume that $(m, l, n) \in \{(m, l, n)|$ no (m_1, l_1, m) nor (m_1, l_1, n) occurs in $S_1\}$, since otherwise $(m, l, n) \in E_I^{max}(S_1) \cap E_I^{min}(S_1)$. By consequence we know

that $(m,l,n) \in \{(m,l,n)|$ no (m_1,l_1,m) nor (m_1,l_1,n) occurs in $S_1.S_2\}$, resulting in $(m,l,n) \in E_I^{max}(S_1.S_2)$. Hence if $(m,l,n) \in E_I^{max}(S_1) \cap (E_I^{min}(S_1) \cup E_I^{max}(S_2))$ then $(m,l,n) \in E_I^{max}(S_1) \cap E_I^{min}(S_1)$ or $(m,l,n) \in E_I^{max}(S_1) \cap E_I^{max}(S_2)$ and hence $(m,l,n) \in E_I^{max}(S_1.S_2)$.

■

This lemma can be generalized to:

Lemma 12 *Let $S_1, S_2, ..., S_n$ and $S_1.S_2...S_n$ be $(n+1)$ correct QL schedules. Then*

$$\begin{array}{lcl} N_I^{min}(S_1...S_n) & = & \bigcup_{i=1}^{n}(N_I^{min}(S_i) \cap (\bigcap_{k<i} N_I^{max}(S_k))) \\ N_I^{max}(S_1...S_n) & = & \bigcap_{i=1}^{n}(N_I^{max}(S_i) \cup (\bigcup_{k<i} N_I^{min}(S_k))) \\ E_I^{min}(S_1...S_n) & = & \bigcup_{i=1}^{n}(E_I^{min}(S_i) \cap (\bigcap_{k<i} E_I^{max}(S_k))) \\ E_I^{max}(S_1...S_n) & = & \bigcap_{i=1}^{n}(E_I^{max}(S_i) \cup (\bigcup_{k<i} E_I^{min}(S_k))) \end{array}$$

If n_a is the number of actions in $S_1.S_2...S_n$ then these equalities can be verified in $O(n_a^3)$-time ($O(n_S^3)$-time) and $O(n_a)$-space ($O(n_S)$-space).

Proof We prove this lemma by induction on n. If $n = 1$ then the four equations are trivial. Suppose that the four equations hold for all $n < N$ (induction hypothesis). If $n = N$, then we know that $S_2, S_3, ..., S_n, S_2.S_3...S_n$ are $N-1$ correct QL schedules, so we can use the induction hypothesis and Lemma 12 to obtain following equations for the N-sets:

$$\begin{array}{lcl} N_I^{min}(S_1...S_n) & = & N_I^{min}(S_1.(S_2...S_n)) \\ & = & N_I^{min}(S_1) \cup (N_I^{max}(S_1) \cap N_I^{min}(S_2...S_n)) \\ & = & N_I^{min}(S_1) \cup (N_I^{max}(S_1) \cap \bigcup_{i=2}^{n}(N_I^{min}(S_i) \cap (\bigcap_{1<k<i} N_I^{max}(S_k)))) \\ & = & N_I^{min}(S_1) \cup (\bigcup_{i=2}^{n}((N_I^{min}(S_i) \cap (\bigcap_{1<k<i} N_I^{max}(S_k))) \cap N_I^{max}(S_1))) \\ & = & N_I^{min}(S_1) \cup (\bigcup_{i=2}^{n}(N_I^{min}(S_i) \cap (\bigcap_{k<i} N_I^{max}(S_k)))) \\ & = & \bigcup_{i=1}^{n}(N_I^{min}(S_i) \cap (\bigcap_{k<i} N_I^{max}(S_k))) \\ N_I^{max}(S_1...S_n) & = & N_I^{max}(S_1.(S_2...S_n)) \\ & = & N_I^{max}(S_1) \cap (N_I^{min}(S_1) \cup N_I^{max}(S_2...S_n)) \\ & = & N_I^{max}(S_1) \cap (N_I^{min}(S_1) \cup \bigcap_{i=2}^{n}(N_I^{max}(S_i) \cup (\bigcup_{1<k<i} N_I^{min}(S_k)))) \\ & = & N_I^{max}(S_1) \cap (\bigcap_{i=2}^{n}((N_I^{max}(S_i) \cup (\bigcup_{1<k<i} N_I^{min}(S_k))) \cup N_I^{min}(S_1))) \\ & = & N_I^{max}(S_1) \cap (\bigcap_{i=2}^{n}(N_I^{max}(S_i) \cup (\bigcup_{k<i} N_I^{min}(S_k)))) \\ & = & \bigcap_{i=1}^{n}(N_I^{max}(S_i) \cup (\bigcup_{k<i} N_I^{min}(S_k))) \end{array}$$

Analogous results are obtained for the E-sets.

The result for these sets can be calculated by following algorithm (this example illustrates N_I^{min}, analogous algorithms exist for the other sets):

```
1  proc N_I^min(S_1...S_n)
2     result_1 := ∅
3     for i := 1 to n
4         result_2 := ∅
5         foreach node ∈ N_I^min(S_i)
6            addNode := true
7            for k := 1 to i − 1
8                if node ∉ N_I^max(S_k)
9                   then addNode := false
10               fi
```

11 **end**
12 **if** $addNode$
13 **then** $result_2 := result_2 \cup \{node\}$
14 **fi**
15 **end**
16 $result_1 := result_1 \cup result_2$
17 **end**
18 $N_I^{min}(S_1...S_n) := result_1$

We see that this algorithm only uses $O(n_a)$-space, since there are a constant number of variables, with $result_1$ and $result_2$ ($O(n_a)$-space) being the largest ones. In order to determine the time complexity, we will use the auxiliary notation $n_{a,i}$ to denote the number of actions in schedule S_i. Since n_a is the number of actions in the combined schedule, $n_a = \sum_{i=1}^{n} n_{a,i}$. On line (8) we will determine whether a node is in a N-set, which can be decided in $O(n_{a,k})$ (this follows from Lemma 7) and hence in $O(n_a)$. Line (9) takes constant time, so (7)-(11) has time complexity $O(n_a^2)$ (since $O(n_a)$ is an upperbound for i). Lines (6) and (12)-(14) take constant time, hence (5)-(15) has time complexity $O(n_{a,i}.n_a^2 + (n_{a,i}.log(n_{a,i})) = O(n_{a,i}.n_a^2)$. The steps at line (4) and (16) take less time, so we will not have to consider them in order to get the time complexity of (3)-(17). Determining the time complexity of this loop results in $\sum_{i=1}^{n} O(n_{a,i}.n_a^2) = \sum_{i=1}^{n} O(n_{a,i}).O(n_a^2) = O(n_a).O(n_a^2) = O(n_a^3)$. From these results follows the total complexity of $O(n_a^3)$-time ($O(n_S^3)$-time) and $O(n_a)$-space ($O(n_S)$-space). ∎

4.3 Deciding Serializability

In this section, we will use the previous theorems to show that serializability is decidable.

Theorem 3 *Given a QL schedule S of n_t transactions and n_a actions. It is decidable whether S is serializable in $O(f(n_t).n_a^3)$-time ($O(f(n_t).n_S^3)$-time), where $f(n_t)$ is exponential in n_t, and in $O(n_a^2)$-space ($O(n_S^2)$-space).*

Proof Deciding whether S is serializable is done in three steps:

- First we verify whether each transaction is correct. For each transaction this is done in $O(n_a^3)$-time and $O(n_a)$-space according to Corollary 1. We can reuse the space in which we decided for each transaction whether it was correct, but obviously we cannot reuse the time, so the total space needed is $O(n_a)$ and the total time needed is $O(n_a^3.n_t)$.

- Then we draw a graph that indicates which transactions can follow directly which other transactions. In order to do this we have to consider all n_t^2 couples of transactions t_i and t_j and decide whether $t_i.t_j$ is correct, which can be done in $O(n_a.log(n_a))$-time and $O(n_a)$-space according to Lemma 10. Since we have to check the correctness of the concatenation for each of the couples transactions we need $O(n_t^2.n_a.log(n_a))$-time. The space requirement is of size $O(n_t^2 + n_a) \subseteq O(n_a^2)$ since we need the n_t^2 results of the correctness and for each check for correctness we can reuse the work space.

- We now see that S is serializable iff there is a Hamiltonian path that is equivalent with S (i.e., the execution of the schedule is equivalent with the execution of some sequence of all transactions of the schedule). We will verify this as follows:

– We first calculate for each of the n_t transactions the ordered N_I^{min}, N_I^{max},E_I^{min} and E_I^{max} sets, which is done in $O(n_t.n_a.log(n_a))$-time and $O(n_t.n_a)$-space (this follows from Lemma 6).
– Then we compute all Hamiltonian paths and check whether it is equivalent with S. For this we use the fact that a Hamiltonian path is a permutation of all n_t nodes.

  1. Compute the first permutation of all n_t nodes.
  2. If the permutation is not a path in the graph then go to (6), else it is a Hamiltonian path. Checking whether the permutation is a path in the graph can be done in $O(n_t)$-time and $O(1)$-space.
  3. We verify the correctness of the Hamiltonian path, which can be done in $O(n_a^3)$-time and $O(n_a)$-space (Corollary 1). If the Hamiltonian path is not correct, then go to (6).
  4. We calculate the ordered N_I^{min}, N_I^{max}, E_I^{min} and E_I^{max} sets of the Hamiltonian path, which is done in $O(n_a.log(n_a))$-time and $O(n_a)$-space (Lemma 6).
  5. To decide whether the Hamiltonian path and S are equivalent we use the results of Theorem 2. But therefore we first have to calculate the set of basic-input-trees which can be done in $O(n_a^3)$-time and in $O(n_a)$-space according to Lemma 12. After the computation of these sets the equivalence can be checked in $O(n_a.log(n_a))$-time and $O(n_a)$-space. If the Hamiltonian path is equivalent with S then S is serializable.
  6. Compute the next permutation of all n_t nodes.

  We know that all permutations can be generated in $O(n_t)$-space and $O(n_t^{n_t})$-time, which is in $O(2^{n_t^2}) = O(f(n_t))$ (we assume $f(n_t) = 2^{n_t^2}$, which is exponential in n_t). Also at most $O(n!) \subseteq O(f(n_t))$ permutations were checked. If we never concluded that S is serializable then S is not serializable.

Hence this step takes $O(f(n_t).n_a^3)$-time ($O(f(n_t).n_S^3)$-time) and $O(n_a^2)$-space ($O(n_a^2)$-space), since for each Hamiltonian path we can reuse the work memory and we have to remember all Hamiltonian paths. This is also the complexity for deciding whether a given QL schedule S is serializable.

■

Chapter 5

Equivalence and Serializability of Schedules

In the previous section we only considered queryless schedules, but in this section we consider all schedules. We start with generalizing the notions that were introduced for QL schedules.

Definition 8 *A schedule S is called correct iff its corresponding QL schedule S' is correct.* $\text{ADD}(S) = \text{ADD}(S')$ *where S' is the QL schedule of S. Analogously for* DEL, E_I^{min}, E_I^{max}, E_O^{min}, E_O^{max}, N_I^{min}, N_I^{max}, N_O^{min}, N_O^{max}.

To verify whether two correct schedules over the same set of transactions are equivalent, we first eliminate the queries and verify whether the resulting QL schedules are equivalent. (Cfr. Theorem 2). In this section we investigate the equivalence of two correct schedules over the same set of transactions and whose QL schedules are equivalent. In the following examples it is shown that such schedules can sometimes be equivalent on all the DTs they are defined on, on only some of them or on none.

Example 6 *Consider the following two schedules:*

$S_1 = (\text{add}(n_2, l_2, n_3), t_1),\ (\text{query}(n_1, l_1/l_2), t_2),$
$(\text{del}(n_2, l_2, n_3), t_1),\ (\text{del}(n_1, l_1, n_2), t_1)$

and

$S_2 = (\text{query}(n_1, l_1/l_2), t_2),\ (\text{add}(n_2, l_2, n_3), t_1),$
$(\text{del}(n_2, l_2, n_3), t_1),\ (\text{del}(n_1, l_1, n_2), t_1)$

S_1 and S_2 are correct and their corresponding QL schedules are equal. They are equivalent on no DT *on which they are defined. Let*

$S_3 = (\text{add}(n_2, l_2, n_3), t_1),\ (\text{query}(n_1, l_1/l_2), t_2),$
$(\text{del}(n_2, l_2, n_3), t_1)$

and

$S_4 = (\text{query}(n_1, l_1/l_2), t_2),\ (\text{add}(n_2, l_2, n_3), t_1),$
$(\text{del}(n_2, l_2, n_3), t_1)$

S_3 and S_4 are correct and their corresponding QL schedules are equal. They are equivalent on some DT*'s on which they are defined and not equivalent on others. Let $l_1 \neq l_3$ and*

$$S_5 = (\text{add}(n_2, l_2, n_3), t_1),\ (\text{query}(n_1, l_1/l_2), t_2),\\ (\text{del}(n_2, l_2, n_3), t_1),\ (\text{del}(n_1, l_3, n_2), t_1)$$

and

$$S_6 = (\text{query}(n_1, l_1/l_2), t_2),\ (\text{add}(n_2, l_2, n_3), t_1),\\ (\text{del}(n_2, l_2, n_3), t_1),\ (\text{del}(n_1, l_3, n_2), t_1)$$

S_5 and S_6 are correct and their corresponding QL schedules are equal. They are equivalent on all DT*'s on which they are defined, hence they are equivalent. Finally let $l \neq l_1$ and*

$$S_7 = (\text{query}(n_1, l/l), t_1),\ (\text{del}(n_2, l, n_3), t_2),\\ (\text{add}(n_2, l_1, n_3), t_2)$$

and

$$S_8 = (\text{del}(n_2, l, n_3), t_2),\\ (\text{add}(n_2, l_1, n_3), t_2),\ (\text{query}(n_1, l/l), t_1)$$

S_7 and S_8 are correct and their corresponding QL schedules are equal. They are equivalent on some DT*'s on which they are defined and not equivalent on others.*

In order to prove the decidability of the equivalence of two transactions over the same set of transactions we first define the notion of SOP, Set Of Prefixes in Subsection 5.1 and some additional notation in Subsection 5.2.

5.1 SOP - Set Of Prefixes

Let pe be a path expression and lp a label path. We define the set of non-empty lp-prefixes in pe, denoted as $\text{SOP}(pe)_{lp}$ as a set of path expressions that together represent the set of label paths pe' such that $pe'/lp \in \mathbf{L}(pe)$[1]. For instance $\text{SOP}(b//*)_a = \{b, b//*\}$.

Definition 9 *Let pe be a path expression, lp be a label path and $l \in \mathcal{L}$. The set of non-empty lp-prefixes in pe, denoted as $\text{SOP}(pe)_{lp}$ is defined by*

- $\text{SOP}(pe)_\epsilon = \{pe\}$
- $\text{SOP}(pe/*)_l = \text{SOP}(pe/l)_l = \{pe\}$
- $\text{SOP}(pe//*)_l = \text{SOP}(pe//l)_l = \{pe, pe//*\}$
- $\text{SOP}(pe/*)_{lp/l} = \text{SOP}(pe/l)_{lp/l} = \text{SOP}(pe)_{lp}$
- $\text{SOP}(pe//*)_{lp/l} = \text{SOP}(pe//l)_{lp/l} = \text{SOP}(pe)_{lp} \cup \text{SOP}(pe//*)_{lp}$
- *Otherwise* $\text{SOP}(pe)_{lp} = \emptyset$.

Furthermore we define $\mathbf{L}(\text{SOP}(pe)_{lp}) = \bigcup_{pe_i \in \text{SOP}(pe)_{lp}} \mathbf{L}(pe_i)$.

[1] We consider pe/ϵ to be equal to pe.

Lemma 13 $\mathbf{L}(\mathrm{SOP}(pe)_{lp}) = \{lp' \mid lp'/lp \in \mathbf{L}(pe)\}$.

Proof We prove this lemma by induction on the length of the label path lp. Suppose $lp = \epsilon$. From the definition follows that:

$$\begin{array}{rcl} \mathbf{L}(\mathrm{SOP}(pe)_{lp}) & = & \bigcup_{pe_i \in \mathrm{SOP}(pe)_{lp}} \mathbf{L}(pe_i) \\ & = & \bigcup_{pe_i \in \mathrm{SOP}(pe)_{\epsilon}} \mathbf{L}(pe_i) \\ & = & \bigcup_{pe_i \in \{pe\}} \mathbf{L}(pe_i) \\ & = & \mathbf{L}(pe) \end{array}$$

Since $lp'/\epsilon = lp'$, we know that $\{lp' \mid lp'/\epsilon \in \mathbf{L}(pe)\} = \mathbf{L}(pe)$. Hence the lemma holds for label paths of length 0.
Assume that the lemma holds for label paths of length n. Then we will prove that it must also hold for label paths of length $n+1$. Suppose lp/l is a label path of length $n+1$. There are three possibilities for the path expression pe for which we want to determine $\mathbf{L}(\mathrm{SOP}(pe)_{lp})$.

- If $pe = pe'/*$ or $pe = pe'/l$ then the following holds:
  $$\begin{array}{rcl} \mathbf{L}(\mathrm{SOP}(pe'/*)_{lp/l}) & = & \mathbf{L}(\mathrm{SOP}(pe'/l)_{lp/l}) \\ & = & \mathbf{L}(\mathrm{SOP}(pe')_{lp}) \\ & = & \{lp' \mid lp'/lp \in \mathbf{L}(pe')\} \\ & = & \{lp' \mid lp'/(lp/l) \in \mathbf{L}(pe'/l)\} \\ & = & \{lp' \mid lp'/(lp/l) \in \mathbf{L}(pe'/*)\} \end{array}$$
  Hence in both cases $\mathbf{L}(\mathrm{SOP}(pe)_{lp}) = \{lp' \mid lp'/lp \in \mathbf{L}(pe)\}$.

- Else if $pe = pe'//*$ or $pe = pe'//l$ then the following holds:
  $$\begin{array}{rcl} \mathbf{L}(\mathrm{SOP}(pe'//*)_{lp/l}) & = & \mathbf{L}(\mathrm{SOP}(pe'//l)_{lp/l}) \\ & = & \mathbf{L}(\mathrm{SOP}(pe')_{lp} \cup \mathrm{SOP}(pe'//*)_{lp}) \\ & = & \mathbf{L}(\mathrm{SOP}(pe')_{lp}) \cup \mathbf{L}(\mathrm{SOP}(pe'//*)_{lp}) \\ & = & \{lp' \mid lp'/lp \in \mathbf{L}(pe')\} \cup \{lp' \mid lp'/lp \in \mathbf{L}(pe'//*)\} \\ & = & \{lp' \mid lp'/lp \in (\mathbf{L}(pe') \cup \mathbf{L}(pe'//*))\} \\ & = & \{lp' \mid lp'/(lp/l) \in (\mathbf{L}(pe'/l) \cup \mathbf{L}(pe'//*/l))\} \\ & = & \{lp' \mid lp'/(lp/l) \in \mathbf{L}(pe'//l)\} \\ & = & \{lp' \mid lp'/(lp/l) \in \mathbf{L}(pe'//*)\} \end{array}$$
  Hence in both cases $\mathbf{L}(\mathrm{SOP}(pe)_{lp}) = \{lp' \mid lp'/lp \in \mathbf{L}(pe)\}$. Note that the validness of the third equation holds by Definition 9.

- Else $\mathrm{SOP}(pe)_{lp/l} = \emptyset$. From Definition 9 then follows that pe has to end with a label $l' \neq l$. Hence $\{lp' \mid lp'/(lp/l) \in \mathbf{L}(pe)\} = \emptyset$.

Hence in all three cases we get the desired equation, which almost concludes our proof. The only case we did not cover is that of label paths of length 1, but this can be proven analogously to the proof for label pahts of length n. ∎

Example 7

- $\mathrm{SOP}(a/*/*/b)_{a/b} = \mathrm{SOP}(a/*/*)_a = \{a/*\}$
- $\mathrm{SOP}(a//*/c)_{a/b/c} = \mathrm{SOP}(a//*)_{a/b} = \mathrm{SOP}(a)_a \cup \mathrm{SOP}(a//*)_a = \{a, a//*\}$
- $\mathrm{SOP}(*//*)_{a/b/c} = \mathrm{SOP}(*)_{a/b} \cup \mathrm{SOP}(*//*)_{a/b} = \emptyset \cup \mathrm{SOP}(*)_a \cup \mathrm{SOP}(*//*)_a = \emptyset \cup \emptyset \cup \{*, *//*\} = \{*, *//*\}$

- $\text{SOP}(a//b//d)_{b/c/d} = \{a, a//*, a//b, a//b//*\}$

Lemma 14 *Let pe be a path expression and lp be a label path.* $\text{SOP}(pe)_{lp} = \{pe' \mid pe' \text{ a prefix of } pe \text{ and } \mathbf{L}(pe'/lp) \subseteq \mathbf{L}(pe)\} \cup \{pe'//* \mid pe' \text{ a prefix of } pe \text{ and } \mathbf{L}(pe'//*/lp) \subseteq \mathbf{L}(pe)\}$. ■

Proof We prove this lemma by induction on the length of the label path lp. For each length of lp we must prove three inclusions, i.e.:

- $\text{SOP}(pe)_{lp} \subseteq \{pe' \mid pe' \text{ a prefix of } pe \text{ and } \mathbf{L}(pe'/lp) \subseteq \mathbf{L}(pe)\} \cup \{pe'//* \mid pe' \text{ a prefix of } pe \text{ and } \mathbf{L}(pe'//*/lp) \subseteq \mathbf{L}(pe)\}$
- $\{pe' \mid pe' \text{ a prefix of } pe \text{ and } \mathbf{L}(pe'/lp) \subseteq \mathbf{L}(pe)\} \subseteq \text{SOP}(pe)_{lp}$
- $\{pe'//* \mid pe' \text{ a prefix of } pe \text{ and } \mathbf{L}(pe'//*/lp) \subseteq \mathbf{L}(pe)\} \subseteq \text{SOP}(pe)_{lp}$

Using these three inclusions we get the equality that need to be proven.

Suppose $lp = \epsilon$

- If $pe' \in \text{SOP}(pe)_{lp}$ then $pe' = pe$ and hence $pe' \in \{pe' \mid pe' \text{ a prefix of } pe \text{ and } \mathbf{L}(pe'/lp) \subseteq \mathbf{L}(pe)\}$.
- If pe' is a prefix of pe and $\mathbf{L}(pe') \subseteq \mathbf{L}(pe)$ then $pe' = pe$ since otherwise there would be label paths in $\mathbf{L}(pe')$ that are not in $\mathbf{L}(pe)$. From $pe = pe'$ follows that $pe' \in \text{SOP}(pe)_\epsilon = \{pe\}$.
- If pe' is a prefix of pe and $\mathbf{L}(pe'//*) \subseteq \mathbf{L}(pe)$ then $pe'//* = pe$ since otherwise there would be label paths in $\mathbf{L}(pe'//*)$ that are not in $\mathbf{L}(pe)$. From $pe = pe'//*$ follows that $pe'//* \in \text{SOP}(pe)_\epsilon = \{pe\}$.

Assume that the Lemma holds for all label paths with length less then N (induction hypothesis). Suppose $lp = lp/l$ and has length $N + 1$.

- Let $pe' \in \text{SOP}(pe)_{lp/l}$. We may assume that pe ends with l or $*$ since otherwise $\text{SOP}(pe)_{lp/l} = \emptyset$. If $pe = pe_2/l$ then $pe' \in \text{SOP}(pe_2)_{lp}$ (Definition 9). From the induction hypothesis follows that pe' is a prefix of pe_2 and $\mathbf{L}(pe'/lp) \subseteq \mathbf{L}(pe_2)$ and hence pe' is a prefix of $pe_2/l = pe$ and $\mathbf{L}(pe'/(lp/l)) \subseteq \mathbf{L}(pe_2/l) = \mathbf{L}(pe)$. All other cases for pe (e.g. $pe = pe_2//l$ or $pe = pe_2/*$) are proven analogously.
- Let pe' be a prefix of pe and $\mathbf{L}(pe'/(lp/l)) \subseteq \mathbf{L}(pe)$. It is obvious that pe has to end with $*$ or l since otherwise the inclusion of the two languages could not hold. If $pe = pe'/pe_2/l$ then $\mathbf{L}(pe'/(lp/l)) \subseteq \mathbf{L}(pe'/pe_2/l)$ and hence $\mathbf{L}(lp/l) \subseteq \mathbf{L}(pe_2/l)$ holds. Furthermore pe' is a prefix of pe'/pe_2 which is shorter than $pe'/pe_2/l$. Therefore we can use the induction hypothesis to derive that $pe' \in \text{SOP}(pe'/pe_2)_{lp} = pe' \in \text{SOP}(pe'/pe_2/l)_{lp/l} = pe' \in \text{SOP}(pe)_{lp/l}$. All other cases for pe (e.g. $pe = pe'//pe_2/l$ or $pe = pe'//pe_2//*$) are proven analogously.
- $\{pe'//* \mid pe' \text{ a prefix of } pe \text{ and } \mathbf{L}(pe'//*/(lp/l)) \subseteq \mathbf{L}(pe)\} \subseteq \text{SOP}(pe)_{lp/l}$ is proven in the same fashion as the previous inclusion.

Note that in fact we also need to proof this lemma for label paths of length 1. This is done analogously to the proof of the induction step. ■

Lemma 15 *Let pe be a path expression of length n_{pe} and lp be a label path of length n_{le}. $\mathrm{SOP}(pe)_{lp}$ is uniquely defined, finite and is computable in $O(n_{pe}^2.(n_{pe}+n_{lp})^2)$-time and $O(log(n_{pe}+n_{lp}))$-space.*

Proof From Lemma 14 we know that we have to calculate the two sets: $\{pe' \mid pe'$ a prefix of pe and $\mathbf{L}(pe'/lp) \subseteq \mathbf{L}(pe)\}$ and $\{pe'//* \mid pe'$ a prefix of pe and $\mathbf{L}(pe'//*/lp) \subseteq \mathbf{L}(pe)\}$. Hence to compute each set we have to check for each prefix pe' of pe (there are n_{pe} such prefixes) whether one path expression contains another path expression (i.e., they have the same language). From the results of [2] we know that we can decide the containment of two path expressions (i.e., does $L(p) \subseteq L(p')$ hold?) in $O(|p|^2|p'|)$-time if p' only contains labels, /, // and $*$. Since this is true, we can decide the containments for a given prefix pe' during the computation of one of the two sets in $O((n_{pe}+n_{lp})^2.n_{pe})$-time ($|p'| = n_{pe}$ and $O(|p|) = O(n_{pe}+n_{lp})$). Hence $\mathrm{SOP}(pe)_{lp}$ can be calculated in $O(n_{pe}^2.(n_{pe}+n_{lp})^2)$-time, and in $O(log(n_{pe}+n_{lp}))$-space (since we only need a pointer for both path expressions). ∎

Lemma 16 *Let pe be a path expression and lp_1 and lp_2 be two label paths. $\mathbf{L}(\mathrm{SOP}(pe)_{lp_1}) \subseteq \mathbf{L}(\mathrm{SOP}(pe)_{lp_2})$ iff $(\forall pe_i \in \mathrm{SOP}(pe)_{lp_1}).(\mathbf{L}(pe_i/lp_2) \subseteq \mathbf{L}(pe))$.*

Proof We prove this lemma by proving two implications

- If $\mathbf{L}(\mathrm{SOP}(pe)_{lp_1}) \subseteq \mathbf{L}(\mathrm{SOP}(pe)_{lp_2})$ then it follows from Lemma 13 that $\mathbf{L}(\mathrm{SOP}(pe)_{lp_1}) \subseteq \{lp' | lp'/lp_2 \in \mathbf{L}(pe)\}$. Hence for all $lp' \in \mathbf{L}(\mathrm{SOP}(pe)_{lp_1})$ it holds that $(lp'/lp_2) \in \mathbf{L}(pe)$. This is equal to saying that $(\forall pe_i \in \mathrm{SOP}(pe)_{lp_1}).(\mathbf{L}(pe_i/lp_2) \subseteq \mathbf{L}(pe))$.
- Suppose $(\forall pe_i \in \mathrm{SOP}(pe)_{lp_1}).(\mathbf{L}(pe_i/lp_2) \subseteq \mathbf{L}(pe))$. If we know for pe_i that $\mathbf{L}(pe_i/lp_2) \subseteq \mathbf{L}(pe)$ then all label paths in $\mathbf{L}(pe_i/lp_2)$ have to be in $\mathbf{L}(pe)$. Hence for all $pe_i \in \mathrm{SOP}(pe)_{lp_1}$ it holds that if $lp' \in \mathbf{L}(pe_i)$ then $lp'/lp_2 \in \mathbf{L}(pe)$. This is equal to saying that if $lp' \in \bigcup_{pe_i \in \mathrm{SOP}(pe)_{lp_1}} \mathbf{L}(pe_i)$ then $lp'/lp_2 \in \mathbf{L}(pe)$. It then follows from Definition 9 and Lemma 13 that if $lp' \in \mathbf{L}(\mathrm{SOP}(pe)_{lp_1})$ then $lp' \in \mathbf{L}(\mathrm{SOP}(pe)_{lp_2})$.

∎

Theorem 4 *Let pe be a path expression and lp_1 and lp_2 be two label paths. If n_{pe} is the length of pe and n_{lp} the length of the longest label path lp_1 or lp_2 then it is decidable in $O(n_{pe}^2.(n_{pe}+n_{lp})^2)$-time and in $O(n_{pe}+(log(n_{pe}+n_{lp})))$-space whether $\mathbf{L}(\mathrm{SOP}(pe)_{lp_1}) = \mathbf{L}(\mathrm{SOP}(pe)_{lp_2})$.*

Proof In order to decide whether the equation holds, we need to check whether both inclusions hold. This can be done by using Lemma 16. If we want to check whether $\mathbf{L}(\mathrm{SOP}(pe)_{lp_1}) \subseteq \mathbf{L}(\mathrm{SOP}(pe)_{lp_2})$ holds, we first have to compute $\mathrm{SOP}(pe)_{lp_1}$ and then check for each path expression pe_i in the result whether $\mathbf{L}(pe_i/lp_2) \subseteq \mathbf{L}(pe)$ holds. The computation of $\mathrm{SOP}(pe)_{lp_1}$ can be done in $O(n_{pe}^2.(n_{pe}+n_{lp})^2)$-time and $O(log(n_{pe}+n_{lp}))$-space (Lemma 15). We also know that the number of path expressions in $\mathrm{SOP}(pe)_{lp_1}$ is at most $2.n_{pe}$ (each prefix and each prefix followed by '//*'). Then we check for each element of the result set whether the inclusion $\mathbf{L}(pe_i/lp_2) \subseteq \mathbf{L}(pe)$ holds, which can be done within the same space and time as the computation of $\mathrm{SOP}(pe)_{lp_1}$. Hence deciding the identity of the languages of two sets of prefixes of the same path expression takes $O(n_{pe}^2.(n_{pe}+n_{lp})^2)$-time and can be done in $O(n_{pe}+(log(n_{pe}+n_{lp})))$-space (we need to store $\mathrm{SOP}(pe)_{lp_1}$ and $\mathrm{SOP}(pe)_{lp_2}$ as intermediate result). ∎

5.2 PQRN - Potential Query Result Nodes

In this section, we will give some additional notation of the concepts that we will need in the next section. We also give some complexity results for calculating the value of these concepts. Let S be a correct schedule that contains the query $Q = \text{query}(n, pe)$.

- We denote by S^Q the actions of S that occur before Q; S^Q is actually a subtransaction of S;
- Let T be a basic-input-tree of S. We define $T^Q = S^Q[T]$ as the DT on which Q in S is evaluated; hence the result of the application of the query Q in S is $Q[T^Q]$;
- We denote by $E^{min}(S^Q)$ as the set that contains exactly those edges that **are required** in T^Q; This set is equal to $(E_I^{min}(S) - \text{DEL}(S^Q)) \cup \text{ADD}(S^Q)$ (Lemma 8);
- We denote by $E^{max}(S^Q)$ as the set that contains exactly those edges that **are allowed** in T^Q; This set is equal to $(E_I^{max}(S) - \text{DEL}(S^Q)) \cup \text{ADD}(S^Q)$ (Lemma 8).

$E^{min}(S^Q)$ is a forest (Property 1). As such every node m of $E^{min}(S^Q)$ has a unique ancestor without a parent in $E^{min}(S^Q)$; it is denoted by $ARoot(S^Q, m)$. The label of the path from $ARoot(S^Q, m)$ to m in $E^{min}(S^Q)$ is denoted by $ALabel(S^Q, m)$.

Lemma 17 *$ALabel(S^Q, m)$ and $ARoot(S^Q, m)$ can be computed in $O(n_a^2)$-time ($O(n_S^2)$-time) and $O(n_a)$-space ($O(n_S)$-space), n_a being the number of actions in S.*

Proof We will compute *ALabel* and *ARoot* together. In order to do this, we first have to compute $E^{min}(S^Q)$, which can be done in $O(n_a.log(n_a))$-space and $O(n_a)$-time (Lemma 6). The result of the computation of $E^{min}(S^Q)$ is a forest with at most $O(n_a)$ edges. Then we execute the following algorithm:

```
1 proc ARoot_and_ALabel(S)
2    n := m;
3    ALabel := m;
4    while ((∃m1, l1).((m1, l1, n) ∈ E^min(S^Q))) do
5          n := m1;
6          ALabel := m1/ALabel
7    end
8    ARoot := n
```

Since there are at most $O(n_a)$ edges in the forest, the while loop will be executed at most $O(n_a)$ times. Each time we have to check the condition $(\exists m_1, l_1).((m_1, l_1, n) \in E^{min}(S^Q))$, which can be done in $O(n_a)$-time and $O(log(n_a))$-space. Hence the total complexity is $O(n_a^2)$-time ($O(n_S^2)$-time) and $O(n_a)$-space ($O(n_S)$-space). ■

If $\text{add}(m, l, n)$ or $\text{del}(m, l, n)$ are operations of S we say that n is a *non-building-node* of S. Otherwise n is called a *building-node* of S. Note that
$E^{max}(S^Q) = E^{min}(S^Q) \cup \{\text{edges that contain only building nodes}\}$ since
$E^{min}(S^Q) = (E_I^{min}(S) - \text{DEL}(S^Q)) \cup \text{ADD}(S^Q)$,
$E^{max}(S^Q) = (E_I^{max}(S) - \text{DEL}(S^Q)) \cup \text{ADD}(S^Q)$ and
$E_I^{max}(S) = E_I^{min}(S) \cup \{\text{edges that contain only building nodes}\}$ (Figure 2).

Lemma 18 *Let S be a correct schedule and T a basic-input-tree of S. Then at any time of the application of S on T all descendants of a non-building-node of S that occurs in the intermediate result T' are also non-building-nodes of S.*

Proof Suppose (m, l, n) is an edge in $T' = S'[T]$, with S' a first part of S, m a non-building-node of S and n a building-node of S. From Lemma 1 follows that $T' = T \cup \text{ADD}(S') - \text{DEL}(S')$. Since $(m, l, n) \in T'$ and n a building-node of S (i.e., no action on an edge ending in n appears in S and hence in S'), we know that (m, l, n) has to be in T. But T is a basic-input-tree and hence it only contains edges that are in $E_I^{max}(S)$. Since m is a non-building-node, we know that an action on an edge ending in m appears in S and hence it follows from Definition 5 that (m, l, n) has to be in $E_I^{min}(S) = \{(m', l', n') \mid \phi_S((m', l', n'), \text{del}(m', l', n'))\}$ and hence $\phi_S((m, l, n), \text{del}(m, l, n))$, which is a contradiction since n is a building-node of S. ∎

We will now show that all edges that occur in $E^{min}(S^Q)$ arrive in non-building-nodes of S and that every non-building-node in $S^Q[T]$ (with T a basic-input-tree of S) is also in $E^{min}(S^Q)$. Therefore we will first introduce a notation for the set of nodes in which the edges of a given set of edges arrive.

Definition 10 *Let E be a set of edges. We define $\Pi_3(E)$ as:*
$\Pi_3(E) = \{n | (\exists m, l).((m, l, n) \in E)\}$.

Lemma 19 *Let S be a correct schedule and Q a query that occurs in S. Then $\Pi_3(E^{min}(S^Q))$ contains only non-building-nodes of S. Furthermore for every non-building-node n of S and for every basic-input-tree T of S, it holds that if n is in $S^Q[T]$ then n is also in $\Pi_3(E^{min}(S^Q))$.*

Proof From the definition it follows that $E^{min}(S^Q) = E_I^{min}(S) - \text{DEL}(S^Q) \cup \text{ADD}(S^Q) = \{(m, l, n) | \phi_S((m, l, n), \text{del}(m, l, n))\} - \text{DEL}(S^Q) \cup \text{ADD}(S^Q)$, so $E^{min}(S^Q)$ contains only edges on which an action in S occurs and hence $\Pi_3(E^{min}(S^Q))$ contains only non-building-nodes of S. On the other hand, suppose that n is a non-building-node, T is a basic-input-tree of S and n is a node of $S^Q[T]$. Then n is a node of $T - \text{DEL}(S^Q) \cup \text{ADD}(S^Q)$ and hence $n \in (N_I^{max}(S) - \Pi_3(\text{DEL}(S^Q))) \cup \Pi_3(\text{ADD}(S^Q))$ (Lemma 1, Definition 3, 4 and 5). Suppose $n \notin \Pi_3(E^{min}(S^Q))$. Then we know that n is not in $\Pi_3(E_I^{min}(S) - \text{DEL}(S^Q))$ and n not in $\Pi_3(\text{ADD}(S^Q))$. Hence $n \in (N_I^{max}(S) - \Pi_3(\text{DEL}(S^Q)))$, which implies that the first occurence of n in S is not its addition (Definition 5) and $n \notin \Pi_3(\text{DEL}(S^Q))$. Since an action occurs on an edge ending in n (n is a non-building-node), we know that this first action on n has to be the deletion of n and hence $n \in \Pi_3(E_I^{min}(S))$. We then get a contradiction with the assumption that $n \notin \Pi_3(E^{min}(S^Q))$ (this follows from the definition of $E^{min}(S^Q)$ and the fact that we have shown that $n \notin \Pi_3(\text{DEL}(S^Q))$ and $n \notin \Pi_3(\text{ADD}(S^Q))$). ∎

We will now define the set of nodes $\text{PQRN}(S, Q)$. This set will contain all non-building-nodes that can be in the result of a query that starts with a node n that is not in $E^{min}(S^Q)$. After the formal definition we will show that this definition corresponds to this informal description. Finally we will show that this set is computable in polynomial time and space.

Definition 11 *Let S be a correct schedule that contains a query $Q = \text{query}(n, pe)$. We define the set $\text{PQRN}(S, Q)$ as:*
$\text{PQRN}(S, Q) = \{m|$

- $m \in E^{min}(S^Q)$;
- m *a non-building-node;*
- $ARoot(S^Q, m)$ *a building-node;*
- $ARoot(S^Q, m) \neq n$;
- $\mathbf{L}(\text{SOP}(pe)_{ALabel(S^Q,m)}) \neq \emptyset$

}.

Lemma 20 *Let S be a correct schedule, $Q = \text{query}(n, pe)$ a query that appears in S and $n \notin E^{min}(S^Q)$. Then $\text{PQRN}(S, Q)$ is the set of non-building-nodes m, such that there exists a basic-input-tree T of S for which m is in the result of the query Q on the document tree $S^Q[T]$.*

Proof We will prove this lemma by proving that if a node m can occur in the result of the query Q on a document tree $S^Q[T]$, then $m \in \text{PQRN}(S, Q)$ and vice versa.

- Assume that m is a non-building-node that can occur in the result of the query Q and $m \notin \text{PQRN}(S, Q)$. Suppose that lp is the label path from n to m in $S^Q[T]$. Since m is a non-building-node and m appears in $S^Q[T]$ we know by Lemma 19 that $m \in \Pi_3(E^{min}(S^Q))$ and hence $m \in E^{min}(S^Q)$. We also know that $ARoot(S^Q, m)$ is a building-node, since $ARoot(S^Q, m) \in S^Q[T]$ and $ARoot(S^Q, m) \notin \Pi_3(E^{min}(S^Q))$ (Lemma 19). Furthermore $ARoot(S^Q, m) \neq n$ because $n \notin E^{min}(S^Q)$. Since we supposed that $m \notin \text{PQRN}(S, Q)$, we now have to prove that $\mathbf{L}(\text{SOP}(pe)_{ALabel(S^Q,m)}) = \emptyset$. We know from the fact that $S^Q[T]$ is a document-tree, m a descendant of $ARoot(S^Q, m)$ and Lemma 19 that the path from n to m has to go through the node $ARoot(S^Q, m)$. For each $S^Q[T]$ the path from $ARoot(S^Q, m)$ to m is fixed and is $ALabel(S^Q, m)$. Since there is a label path lp from n to m such that $lp \in \mathbf{L}(pe)$ (m can be in the result of $\text{query}(n, pe)$) and lp always end with $ALabel(S^Q, m)$ (i.e., $lp = lp'/ALabel(S^Q, m)$ for a given lp'), it follows from Lemma 13 that there is a $lp' \in \mathbf{L}(\text{SOP}(pe)_{ALabel(S^Q,m)})$, which is a contradiction with our assumption that $\mathbf{L}(\text{SOP}(pe)_{ALabel(S^Q,m)})$ is empty.

- Assume $m \in \text{PQRN}(S, Q)$. From Lemma 19 and the assumption that $n \notin E^{min}(S^Q)$ we know that n is a building node. Hence it follows from $ARoot(S^Q, m) \neq n$ that for each label path lp there is a corresponding path in $E_I^{max}(S)$ from n to $ARoot(S^Q, m)$ with the same labels. We also know that m is always in the document tree on which the query Q gets applied in S since $m \in E^{min}(S^Q)$. From the fact that $\mathbf{L}(\text{SOP}(pe)_{ALabel(S^Q,m)}) \neq \emptyset\}$ then follows that there is a label path that we can use for a path in $E^{max}(S^Q)$ from n to $ARoot(S^Q, m)$ such that m is in the result of Q. Hence there is a basic-input-tree T such that Q applied on $S^Q[T]$ has the node m in its result set.

■

Lemma 21 $\text{PQRN}(S, Q)$ *can be computed in $O(n_a(n_a^2 + n_{pe}^2.(n_{pe} + n_a)^2))$-time ($O(n_S^5)$-time) and $O(n_a + n_{pe})$-space ($O(n_S)$-space).*

Proof We start the computation of PQRN(S, Q) by computing $E^{min}(S^Q)$. This can be done in $O(n_a.log(n_a))$-time and $O(n_a)$-space. Then we check for every node in $E^{min}(S^Q)$ whether it is a non-building-node and we make a set of all non-building-nodes that appear in edges of $E^{min}(S^Q)$. We need $O(n_a^2)$-time to make this set and $O(n_a)$-space to save the result as an intermediate result. For each node in the intermediate result, which has a size of $O(n_a)$, we compute $ARoot(S^Q, m)$ and if this root is equal to n or a non-building-node then we remove the node from the intermediate result. The computation of $ARoot(S^Q, m)$ can be done in $O(n_a^2)$-time and $O(n_a)$-space (Lemma 17). Checking whether the root is a non-building-node and comparing the root to n needs less time and space. Until now we needed $O(n_a(n_a^2))$-time and $O(n_a)$-space. All what is left, is checking whether $\mathbf{L}(\text{SOP}(pe)_{ALabel(S^Q,m)}) \neq \emptyset$ holds. From Definition 9 follows that $\mathbf{L}(\text{SOP}(pe)_{ALabel(S^Q,m)})$ is empty iff $\text{SOP}(pe)_{ALabel(S^Q,m)}$ is empty or $\mathbf{L}(pe_i)$ is empty for all $pe_i \in \text{SOP}(pe)_{ALabel(S^Q,m)}$. Since we know from the definition that the language off a path expression is never empty, it suffices to check whether $\text{SOP}(pe)_{ALabel(S^Q,m)}$ is empty. We do this by computing $\text{SOP}(pe)_{ALabel(S^Q,m)}$. According to Lemma 15 we need $O(n_{pe}^2.(n_{pe}+n_a)^2)$-time and $O(n_{pe}+(log(n_{pe}+n_a)))$-space to do this, since $n_{lp} = O(n_a)$. Hence the computation of PQRN(S, Q) can be done in $O(n_a(n_a^2 + n_{pe}^2.(n_{pe} + n_a)^2))$-time and $O(n_a + n_{pe})$-space. Since both n_a and n_{pe} are in $O(n_S)$ we are allowed to rewrite these results to a complexity of $O(n_S^5)$-time and $O(n_S)$-space. ■

5.3 Deciding Serializability

In order to decide whether a query Q gives the same answer in two correct schedules S_1 and S_2 for any basic-input-tree on which they are defined, given that their QL schedules are equivalent, we need to define a condition $CND(S_1, S_2, Q)$ and prove that this condition in deed detects when Q gives the same answer in S_1 as in S_2 for every basic-input-tree of S_1 and S_2.

Definition 12 *We define the condition $CND(S_1, S_2, Q)$ as:*

1. $\{m \mid$ *there is a path of* $\mathbf{L}$*(pe) from n to m in* $E^{min}(S_1^Q)\} = \{m \mid$ *there is a path of* $\mathbf{L}$*(pe) from n to m in* $E^{min}(S_2^Q)\}$*;*
2. *furthermore, if n* **is a building-node of** *S_i:*
   - *(a)* $\text{PQRN}(S_1, Q) = \text{PQRN}(S_2, Q)$
   - *(b) for the nodes $m \in \text{PQRN}(S_1, Q)$ hold that*
     - *i.* $ARoot(S_1^Q, m) = ARoot(S_2^Q, m)$
     - *ii.* $\mathbf{L}(\text{SOP}(pe)_{ALabel(S_1^Q,m)}) = \mathbf{L}(\text{SOP}(pe)_{ALabel(S_2^Q,m)})$

Lemma 22 *Given two correct schedules S_1 and S_2 over the same set of transactions and whose QL schedules are equivalent. Then Q gives the same answer in S_1 as in S_2 for every possible basic-input-tree of S_1 and S_2 iff $CND(S_1, S_2, Q)$ holds.*

Proof We prove this lemma in two steps: first we show that if $CND(S_1, S_2, Q)$ then $Q = query(n, pe)$ gives the same answer in S_1 as in S_2 for every possible basic-input-tree of S_1 and S_2, and then we show the reverse.

- Suppose that $CND(S_1, S_2, Q)$ holds. Then we consider the four cases for n in which it is or is not in $\Pi_3(E^{min}(S_1^Q))$ and/or $\Pi_3(E^{min}(S_2^Q))$.

  1. If $n \in \Pi_3(E^{min}(S_1^Q))$ and $n \in \Pi_3(E^{min}(S_2^Q))$ then we know that n is a non-building-node of S (Lemma 19) and hence all descendants of n in $S_1^Q[T]$ and $S_2^Q[T]$ are also non-building-nodes of S (Lemma 18). From this follows, by applying Lemma 19, that these descendants are also in respectively $E^{min}(S_1^Q)$ and $n \in E^{min}(S_2^Q)$. For this reason and since we only allow downward axes in our queries we know that the result of the query is always in $E^{min}(S_i^Q)$ if $n \in \Pi_3(E^{min}(S_i^Q))$ for $i = 1, 2$. Hence we know by the first condition of CND that the query Q has always the same result in S_1 as in S_2 for any basic-input-tree of S_1 and S_2.
  2. Suppose $n \in \Pi_3(E^{min}(S_1^Q))$ and $n \notin \Pi_3(E^{min}(S_2^Q))$. Since n is a non-building-node of S_1 (Lemma 19) and the schedules S_1 and S_2 are over the same set of transactions, we know that n is also a non-building-node of S_2. But from Lemma 19 it then follows $(\forall T).(n \in S_2^Q[T] \rightarrow n \in \Pi_3(E^{min}(S_2^Q)))$ and hence $(\forall T).(n \notin S_2^Q[T])$. From this follows that the result of the query in S_2 is always empty and hence $\{m \mid$ there is a path of $\mathbf{L}(\text{pe})$ from n to m in $E^{min}(S_1^Q)\}$ is also empty. From the first part of the CND condition we then know that $\{m \mid$ there is a path of $\mathbf{L}(\text{pe})$ from n to m in $E^{min}(S_1^Q)\}$ is empty. It follows that also the result of Q in S_1 is always empty, since $n \in \Pi_3(E^{min}(S_1^Q))$ and since we know that, as in the previous item, the result of Q in S_1 is always in $E^{min}(S_1^Q)$.
  3. The case for $n \notin \Pi_3(E^{min}(S_1^Q))$ and $n \in \Pi_3(E^{min}(S_2^Q))$ is analogous to the previous case.
  4. Suppose $n \notin \Pi_3(E^{min}(S_1^Q))$ and $n \notin \Pi_3(E^{min}(S_2^Q))$. If n is a non-building-node of S_1 and S_2 then we have seen in the previous parts of this proof that the result of Q in both S_1 and S_2 is always empty for every basic-input-tree of S_1 and S_2. Therefore suppose that n is a building-node of S_1 and S_2. Then any building-node m is a node in a basic-input-tree T iff $m \in S_1^Q[T]$ and $m \in S_2^Q[T]$ (since m does not get deleted nor added in S_1 and S_2). Hence it is impossible for building-nodes to be in the result of Q for S_1 and not in the result of Q for S_2 (since it follows from Lemma 18 that no non-building-nodes lie between n and a descendant that is a building-node). For any label path we can construct a path from building-nodes in T (and hence in $S_1^Q[T]$ and $S_2^Q[T]$) from n to a root of $E_I^{min}(S_1)$, since the constructed path is a subset of $E_I^{max}(S_1)$ Note that $E_I^{min}(S_1) = E_I^{min}(S_2)$ and $E_I^{max}(S_1) = E_I^{max}(S_2)$, since the QL schedules are equivalent (Theorem 2). Since the second part of CND tells us that $\text{PQRN}(S_1, Q) = \text{PQRN}(S_2, Q)$, we know that the set of nodes of $E^{min}(S_1)$ that can be in the result of Q in S_1 is the same as the set of nodes of $E^{min}(S_2)$ that can be in the result of Q in S_2. But this does not mean that for each of these nodes m, the set of basic-input-trees T for which m is in the result of the query is the same for S_1 and S_2. The fact that these sets are the same follows from $2.b$ of the CND condition, which states that m has the same root r in $E^{min}(S_1^Q)$ as in $E^{min}(S_2^Q)$ and that the set of prefixes (SOP) for the label path from r to m in S_1^Q has the same language as that for the SOP for the label path from r to m in S_2^Q. We still have to note that this does not hold when n is a root of $E^{min}(S_1^Q)$ and hence (by the first subcondition of part $2.b$ of the

CND condition) a root of $E^{min}(S_2^Q)$, since we don't have the ability to construct the paths in order to get nodes in the result of the query. But then it follows from the first part of the CND condition that the result of the query Q is the same in S_1 and S_2 for every basic-input-tree.

- Suppose that Q gives the same answer in S_1 as in S_2 for every possible basic-input-tree of S_1 and S_2. We then have to show that both subconditions of $CND(S_1, S_2, Q)$ have to hold.

  1. If $\{m \mid$ there is a path of $\mathbf{L}$(pe) from n to m in $E^{min}(S_1^Q)\} \neq \{m \mid$ there is a path of $\mathbf{L}$(pe) from n to m in $E^{min}(S_2^Q)\}$ then suppose (without loss of generality) that m is a node for which there is a path of $\mathbf{L}(pe)$ from n to m in $E^{min}(S_1^Q)$ and there is no path of $\mathbf{L}(pe)$ from n to m in $E^{min}(S_1^Q)$, then m is always in the result of Q in S_1 but is not always in the result of Q in S_2. Since S_1 and S_2 are equivalent, they are defined on the same set of basic-input-trees (Theorem 2), hence there is a basic-input-tree for which Q gives another answer in S_1 than in S_2 for Q.
  2. Assume n is a building-node of S_i.
     (a) Suppose $\text{PQRN}(S_1, Q) \neq \text{PQRN}(S_2, Q)$. Then we may assume (without loss of generality) that $m \in \text{PQRN}(S_1, Q)$ and $m \notin \text{PQRN}(S_2, Q)$. We now know by Lemma 20 that there exists a basic-input-tree T such that m is in the result of query Q when applied to $S_1^Q[T]$ and for every basic-input-tree T it holds that m is not in the result of query Q when applied to $S_2^Q[T]$. This is a contradiction with our assumption that the result of Q in S_1 and S_2 is the same for any basic-input-tree.
     (b) We now show for the nodes $m \in \text{PQRN}(S_1, Q) = \text{PQRN}(S_2, Q)$ that the two subconditions of part 2.b of the CND condition have to hold:
        i. Suppose $ARoot(S_1^Q, m) \neq ARoot(S_2^Q, m)$. Since $m \in \text{PQRN}(S_1, Q)$, we know that we can construct a basic-input-tree T with a label path lp from node n to $ARoot(S_1^Q, m)$ such that m is in the result of Q applied on $S_1^Q[T]$ (Lemma 20). Since $ARoot(S_2^Q, m)$ is a building-node that is not equal to $ARoot(S_1^Q, m)$, we are allowed to add an edge from another building-node to $ARoot(S_1^Q, m)$ and n such that n and $ARoot(S_1^Q, m)$ are siblings. Since we only consider downward axes in our path expressions, no descendant of $ARoot(S_2^Q, m)$ can be in the result of the query Q on this document-tree. Hence m is in the result for Q in $S_1^Q[T]$ but not in $S_2^Q[T]$.
        ii. Suppose $\mathbf{L}(\text{SOP}(pe)_{ALabel(S_1^Q, m)}) \neq \mathbf{L}(\text{SOP}(pe)_{ALabel(S_2^Q, m)})$. Then suppose (without loss of generality) that the label path lp is in the language $\mathbf{L}(\text{SOP}(pe)_{ALabel(S_1^Q, m)})$ but not in $\mathbf{L}(\text{SOP}(pe)_{ALabel(S_2^Q, m)})$. This means that we can construct a path in a basic-input-tree T with corresponding labels lp from n to $r = ARoot(S_1^Q, m)$ such that $lp/ALabel(S_1^Q, m) \in \mathbf{L}(pe)$ (Lemma 13) and hence m is in the result of the query Q for $S_1^Q[T]$. Since $lp \notin \mathbf{L}(\text{SOP}(pe)_{ALabel(S_2^Q, m)})$, it follows that m is not in the result of the query Q for $S_2^Q[T]$ (since by Lemma 13 we know that $lp/ALabel(S_2^Q, m) \notin \mathbf{L}(pe)$).

     Hence also the second subcondition of CND has to hold.

This concludes the proof of 'correctness' of CND.

■

Lemma 23 *Given two correct schedules S_1 and S_2 over the same set of transactions and whose QL schedules are equivalent. Let $Q = \text{query}(n, pe)$ be a query in these schedules and let n_a be the total number of actions in S_1 and S_2. It is decidable in $O(n_a(n_a^2 + n_{pe}^2.(n_{pe} + n_a)^2))$-time ($O(n_S^5)$-time) and $O(n_a + n_{pe})$-space ($O(n_S)$-space) whether Q gives the same answer in S_1 as in S_2 for every possible basic-input-tree of S_1 and S_2.*

Proof From Lemma 22 follows that we have to prove that the we can check the CND condition in $O(n_a(n_a^2 + n_{pe}^2.(n_{pe} + n_a)^2))$-time and $O(n_a + n_{pe})$-space. The first part of the definition is decided in $O(n_a.n_{pe}^2)$-time and $O(log(n_a + n_{pe}))$-space. This is a consequence of a result in [2]. In the second part we first have to compute $\text{PQRN}(S_1, Q)$ and $\text{PQRN}(S_2, Q)$, which can be done in $O(n_a(n_a^2 + n_{pe}^2.(n_{pe} + n_a)^2))$-time and $O(n_a + n_{pe})$-space (Lemma 21). Then we compare those sets, which can be done in less time and space than the computation of PQRN, since the size of the set PQRN is at most $O(n_a)$. For each element in $\text{PQRN}(S_1, Q)$ we check whether the two subconditions hold (i.e. $ARoot(S_1^Q, m) = ARoot(S_2^Q, m)$ and $\mathbf{L}(\text{SOP}(pe)_{ALabel(S_1^Q, m)}) = \mathbf{L}(\text{SOP}(pe)_{ALabel(S_2^Q, m)})$). The first subcondition can be checked in $O(n_a^2)$-time and $O(n_a)$-space (Lemma 17), the second in $O(n_{pe}^2.(n_{pe} + n_a)^2)$-time and in $O(n_{pe} + (log(n_{pe} + n_a)))$-space (Theorem 4). Hence checking the two subconditions for all elements of the PQRN-set can be done in $O(n_a(n_a^2 + n_{pe}^2.(n_{pe} + n_a)^2))$-time ($O(n_S^5)$-time) and $O(n_a + n_{pe})$-space ($O(n_S)$-space). This is also the total time and space complexity for deciding whether CND holds, since the first subcondition was shown to be decidable in $O(n_a(n_a^2 + n_{pe}^2.(n_{pe} + n_a)^2))$-time and $O(n_a + n_{pe})$-space. ■

Example 8 *In Example 6 we have $E^{min}(S_1^Q) = \{(n_1, l_1, n_2), (n_2, l_2, n_3)\}$ and $E^{min}(S_2^Q) = \{(n_1, l_1, n_2)\}$; hence S_1 and S_2 are not equivalent, since 1. is not fulfilled and Q does not give the same answer in S_1 as in S_2 for every possible basic-input-tree of S_1 and S_2.*

$E^{min}(S_3^Q) = \{(n_2, l_2, n_3)\}$ and $E^{min}(S_4^Q) = \emptyset$; 1. is fulfilled; n_2 is a building-node; n_3 is a non-building-node; $\text{PQRN}(S_3, Q) = \{n_3\}$ and $\text{PQRN}(S_4, Q) = \emptyset$; hence 2.(a) is not fulfilled and Q does not give the same answer in S_3 as in S_4 for every possible basic-input-tree of S_3 and S_4.

$E^{min}(S_5^Q) = \{(n_1, l_3, n_2), (n_2, l_2, n_3)\}$ and $E^{min}(S_6^Q) = \{(n_1, l_3, n_2)\}$; 1. is fulfilled; n_1 is a building-node; n_2 and n_3 are non-building-nodes; $\text{PQRN}(S_5, Q) = \text{PQRN}(S_6, Q) = \emptyset$; hence $CND(S_5, S_6, Q)$ is fulfilled and Q gives the same answer in S_5 as in S_6 for every possible basic-input-tree of S_5 and S_6.

$E^{min}(S_7^Q) = \{(n_2, l, n_3)\}$ and $E^{min}(S_8^Q) = \{(n_2, l_1, n_3)\}\}$; 1. is fulfilled; n_2 is a building-node; n_3 is a non-building-node; $\text{PQRN}(S_7, Q) = \text{PQRN}(S_8, Q) = \{n_3\}$; $ARoot(S_7^Q, n_3) = n_2 = ARoot(S_8^Q, n_3)$, but $\text{SOP}(l/l)_l = \{l\} \neq \emptyset = \text{SOP}(l/l)_{l_1}$ hence $CND(S_7, S_8, Q)$ is not fulfilled and Q does not give the same answer in S_7 as in S_8 for every possible basic-input-tree of S_7 and S_8.

Theorem 5 *Given two correct schedules S_1 and S_2 over the same set of transactions and whose QL schedules are equivalent. It is decidable in $O(n_a^2(n_a^2 + n_{pe}^2.(n_{pe} + n_a)^2))$-time ($O(n_S^6)$-time) and $O(n_a + n_{pe})$-space ($O(n_S)$-space) whether they are equivalent, with n_{pe} the length of the longest path expression in a query of S_1 or S_2.*

Proof In order to check the equivalence of two schedules over the same set of transactions we have to check whether their QL schedules are equivalent and whether for each query Q in both schedules we get the same result for every basic-input-tree of S_1 and S_2. The first part can be done in $O(n_a.log(n_a))$-time and $O(n_a)$-space (Theorem 2). Since there are at most $O(n_a)$ queries in both schedules, the second part can be decided in $O(n_a(n_a^2+n_{pe}^2.(n_{pe}+n_a)^2))$-time and $O(n_a+n_{pe})$-space (Lemma 23). This results in a total complexity of $O(n_a^2(n_a^2+n_{pe}^2.(n_{pe}+n_a)^2))$-time ($O(n_S^6)$-time) and $O(n_a+n_{pe})$-space ($O(n_S)$-space). ■

Theorem 6 *Given a correct schedule S. It is decidable in $O(f(n_t).n_a^2(n_a^2+n_{pe}^2.(n_{pe}+n_a)^2))$-time ($O(n_S^6)$-time) and $O(n_a^2+n_{pe})$-space ($O(n_S^2)$-space) whether S is serializable, where $f(n_t)$ is exponential in n_t.*

Proof The proof of this Theorem is analogous to the proof of Theorem 3. The only difference is that we now have to check whether the schedule corresponding to the Hamiltonian path and S are equivalent using a different approach, since we now allow queries in S. The algorithm is the same as in the proof of Theorem 3, except that we now use Theorem 5 in stead of Theorem 2 to check equivalence of the schedules (this is done in the fifth substep of the third step). Since checking the equivalence needs to be done for all Hamiltonian paths we get a total time complexity of $O(f(n_t).n_a^2(n_a^2+n_{pe}^2.(n_{pe}+n_a)^2))$ ($O(n_S^6)$) and we will need $O(n_a^2+n_{pe})$-space ($O(n_S^2)$-space) to decide the serializability. ■

References

[1] S. Abiteboul, P. Buneman, and D. Suciu. *Data on the Web: From Relations to Semistructured Data and XML*. Morgan-Kaufmann, San Francisco, 1999.

[2] G. Miklau and D. Suciu. Containment and equivalence for an XPath fragment. In *Symposium on Principles of Database Systems*, pages 65–76, 2002.